

VxWorks[®]

Reference Manual

5.4

Edition 1

Copyright © 1984 – 1999 Wind River Systems, Inc.

ALL RIGHTS RESERVED. No part of this publication may be copied in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Wind River Systems, Inc.

VxWorks, IxWorks, Wind River Systems, the Wind River Systems logo, *wind*, and Embedded Internet are registered trademarks of Wind River Systems, Inc. Tornado, CrossWind, Personal JWorks, VxMP, VxSim, VxVMI, WindC++, WindConfig, Wind Foundation Classes, WindNet, WindPower, WindSh, and WindView are trademarks of Wind River Systems, Inc.

All other trademarks used in this document are the property of their respective owners.

Corporate Headquarters

Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
USA

toll free (US): 800/545-WIND
telephone: 510/748-4100
facsimile: 510/749-2010

Europe

Wind River Systems, S.A.R.L.
19, Avenue de Norvège
Immeuble B4, Bâtiment 3
Z.A. de Courtaboeuf 1
91953 Les Ulis Cédex
FRANCE

telephone: 33-1-60-92-63-00
facsimile: 33-1-60-92-63-15

Japan

Wind River Systems K.K.
Ebisu Prime Square Tower 5th Fl.
1-1-39 Hiroo
Shibuya-ku
Tokyo 150-0012
JAPAN

telephone: 81-3-5778-6001
facsimile: 81-3-5778-6002

CUSTOMER SUPPORT

| | Telephone | E-mail | Fax |
|-------------------|--|--------------------|--------------------|
| Corporate: | 800/872-4977 toll free, U.S. & Canada 510/748-4100 direct | support@wrs.com | 510/749-2164 |
| Europe: | 33-1-69-07-78-78 | support@wrsec.fr | 33-1-69-07-08-26 |
| Japan: | 011-81-3-5467-5900 | support@kk.wrs.com | 011-81-3-5467-5877 |

If you purchased your Wind River Systems product from a distributor, please contact your distributor to determine how to reach your technical support organization.

Please provide your license number when contacting Customer Support.

Contents

1 Libraries

This section provides reference pages for VxWorks libraries. Each entry lists the routines found in the library, including a one-line synopsis of each and a general description of their use.

Entries for libraries that are specific to board support packages (BSPs) are provided in online format only. However, this section contains entries for the serial, Ethernet, and SCSI drivers available with VxWorks BSPs, plus a generic entry for the BSP-specific library `sysLib`.

2 Subroutines

This section provides reference pages for each of the subroutines found in VxWorks libraries documented in section 1.

Keyword Index

This section is a “permuted index” of keywords found in the NAME line of each reference page. The keyword for each index item is left-aligned in column 2. The remaining words in column 1 and 2 show the context for the keyword.

1

Libraries

| | | |
|------------------------|--|------|
| aic7880Lib | – Adaptec 7880 SCSI Host Adapter Library File | 1-1 |
| aioPxLib | – asynchronous I/O (AIO) library (POSIX) | 1-4 |
| aioPxShow | – asynchronous I/O (AIO) show library | 1-8 |
| aioSysDrv | – AIO system driver | 1-9 |
| ambaSio | – ARM AMBA UART tty driver | 1-9 |
| ansiAssert | – ANSI assert documentation | 1-12 |
| ansiCtype | – ANSI ctype documentation | 1-12 |
| ansiLocale | – ANSI locale documentation | 1-13 |
| ansiMath | – ANSI math documentation | 1-13 |
| ansiSetjmp | – ANSI setjmp documentation | 1-15 |
| ansiStdarg | – ANSI stdarg documentation | 1-15 |
| ansiStdio | – ANSI stdio documentation | 1-16 |
| ansiStdlib | – ANSI stdlib documentation | 1-20 |
| ansiString | – ANSI string documentation | 1-21 |
| ansiTime | – ANSI time documentation | 1-22 |
| arpLib | – Address Resolution Protocol (ARP) table manipulation library | 1-24 |
| ataDrv | – ATA/IDE (LOCAL and PCMCIA) disk device driver | 1-24 |
| ataShow | – ATA/IDE (LOCAL and PCMCIA) disk device driver show routine | 1-27 |
| bALib | – buffer manipulation library SPARC assembly language routines | 1-27 |
| bLib | – buffer manipulation library | 1-28 |
| bootConfig | – system configuration module for boot ROMs | 1-29 |
| bootInit | – ROM initialization module | 1-29 |
| bootLib | – boot ROM subroutine library | 1-30 |
| bootpLib | – BOOTP client library | 1-32 |
| cacheArchLib | – architecture-specific cache management library | 1-34 |
| cacheCy604Lib | – Cypress CY7C604/605 SPARC cache management library | 1-35 |
| cacheI960CxALib | – I960Cx cache management assembly routines | 1-35 |
| cacheI960CxLib | – I960Cx cache management library | 1-36 |
| cacheI960JxALib | – I960Jx cache management assembly routines | 1-36 |
| cacheI960JxLib | – I960Jx cache management library | 1-37 |

| | | |
|--------------------------|--|-------|
| cacheLib | – cache management library | 1-37 |
| cacheMb930Lib | – Fujitsu MB86930 (SPARClite) cache management library | 1-46 |
| cacheMicroSparLib | – microSPARC cache management library | 1-46 |
| cacheR3kALib | – MIPS R3000 cache management assembly routines | 1-47 |
| cacheR3kLib | – MIPS R3000 cache management library | 1-47 |
| cacheR4kLib | – MIPS R4000 cache management library | 1-48 |
| cacheR33kLib | – MIPS R33000 cache management library | 1-48 |
| cacheR333x0Lib | – MIPS R333x0 cache management library | 1-49 |
| cacheSun4Lib | – Sun-4 cache management library | 1-49 |
| cacheTiTms390Lib | – TI TMS390 SuperSPARC cache management library | 1-50 |
| cd2400Sio | – CL-CD2400 MPCC serial driver | 1-52 |
| cdromFsLib | – ISO 9660 CD-ROM read-only file system library | 1-52 |
| cisLib | – PCMCIA CIS library | 1-56 |
| cisShow | – PCMCIA CIS show library | 1-57 |
| clockLib | – clock library (POSIX) | 1-57 |
| cplusLib | – basic run-time support for C++ | 1-58 |
| dbgArchLib | – architecture-dependent debugger library | 1-59 |
| dbgLib | – debugging facilities | 1-60 |
| dec21x4xEnd | – END style DEC 21x4x PCI Ethernet network interface driver | 1-63 |
| dec21x40End | – END-style DEC 21x40 PCI Ethernet network interface driver | 1-67 |
| dhcpcBootLib | – DHCP boot-time client library | 1-71 |
| dhcpcLib | – Dynamic Host Configuration Protocol (DHCP) run-time client API | 1-72 |
| dhcpcShow | – DHCP run-time client information display routines | 1-74 |
| dhcprLib | – DHCP relay agent library | 1-74 |
| dhcpsLib | – Dynamic Host Configuration Protocol (DHCP) server library | 1-75 |
| dirLib | – directory handling library (POSIX) | 1-80 |
| dosFsLib | – MS-DOS media-compatible file system library | 1-82 |
| ei82596End | – END style Intel 82596 Ethernet network interface driver | 1-96 |
| el3c90xEnd | – END network interface driver for 3COM 3C90xB XL | 1-99 |
| elt3c509End | – END network interface driver for 3COM 3C509 | 1-104 |
| endLib | – support library for END-based drivers | 1-106 |
| envLib | – environment variable library | 1-106 |
| errnoLib | – error status library | 1-107 |
| etherLib | – Ethernet raw I/O routines and hooks | 1-109 |
| etherMultiLib | – a library to handle Ethernet multicast addresses | 1-111 |
| evbNs16550Sio | – NS16550 serial driver for the IBM PPC403GA evaluation | 1-111 |
| excArchLib | – architecture-specific exception-handling facilities | 1-112 |
| excLib | – generic exception handling facilities | 1-112 |
| fei82557End | – END style Intel 82557 Ethernet network interface driver | 1-114 |
| fiolib | – formatted I/O library | 1-116 |
| floatLib | – floating-point formatting and scanning library | 1-117 |
| fppArchLib | – architecture-dependent floating-point coprocessor support | 1-118 |
| fppLib | – floating-point coprocessor support library | 1-119 |
| fppShow | – floating-point show routines | 1-120 |
| ftpdLib | – File Transfer Protocol (FTP) server | 1-120 |

| | | |
|---------------------------|---|-------|
| ftpLib | – File Transfer Protocol (FTP) library | 1-121 |
| hostLib | – host table subroutine library | 1-123 |
| i8250Sio | – I8250 serial driver | 1-123 |
| icmpShow | – ICMP Information display routines | 1-124 |
| ideDrv | – IDE disk device driver | 1-125 |
| ifLib | – network interface library | 1-125 |
| if_cpm | – Motorola CPM core network interface driver | 1-126 |
| if_cs | – Crystal Semiconductor CS8900 network interface driver | 1-129 |
| if_dc | – DEC 21x4x Ethernet LAN network interface driver | 1-133 |
| if_eex | – Intel EtherExpress 16 network interface driver | 1-136 |
| if_ei | – Intel 82596 Ethernet network interface driver | 1-137 |
| if_eihk | – Intel 82596 Ethernet network interface driver for hkv3500 | 1-140 |
| if_elc | – SMC 8013WC Ethernet network interface driver | 1-144 |
| if_elt | – 3Com 3C509 Ethernet network interface driver | 1-144 |
| if_ene | – Novell/Eagle NE2000 network interface driver | 1-146 |
| if_esmc | – Ampro Ethernet2 SMC-91c9x Ethernet network interface driver | 1-147 |
| if_fei | – Intel 82557 Ethernet network interface driver | 1-149 |
| if_fn | – Fujitsu MB86960 NICE Ethernet network interface driver | 1-151 |
| if_ln | – AMD Am7990 LANCE Ethernet network interface driver | 1-153 |
| if_lnPci | – AMD Am79C970 PCnet-PCI Ethernet network interface driver | 1-156 |
| if_loop | – software loopback network interface driver | 1-160 |
| if_mbc | – Motorola 68EN302 network-interface driver | 1-160 |
| if_nicEvb | – National Semiconductor ST-NIC Chip network interface driver | 1-163 |
| if_sl | – Serial Line IP (SLIP) network interface driver | 1-164 |
| if_sm | – shared memory backplane network interface driver | 1-166 |
| if_sn | – National Semiconductor DP83932B SONIC Ethernet network driver | 1-167 |
| if_ulip | – network interface driver for User Level IP (VxSim) | 1-170 |
| if_ultra | – SMC Elite Ultra Ethernet network interface driver | 1-171 |
| igmpShow | – IGMP information display routines | 1-171 |
| inetLib | – Internet address manipulation routines | 1-172 |
| inflateLib | – inflate code using public domain zlib functions | 1-173 |
| intArchLib | – architecture-dependent interrupt library | 1-174 |
| intLib | – architecture-independent interrupt subroutine library | 1-175 |
| ioLib | – I/O interface library | 1-176 |
| iOlicomEnd | – END style Intel Olicom PCMCIA network interface driver | 1-177 |
| ioMmuMicroSparcLib | – microSparc I/II I/O DMA library | 1-180 |
| iosLib | – I/O system library | 1-180 |
| iosShow | – I/O system show routines | 1-181 |
| ipFilterLib | – ip filter hooks library | 1-181 |
| ipProto | – an interface between the BSD IP protocol and the MUX | 1-182 |
| kernelLib | – VxWorks kernel library | 1-182 |
| ledLib | – line-editing library | 1-184 |
| ln97xEnd | – END style AMD Am79C97X PCnet-PCI Ethernet driver | 1-186 |
| ln7990End | – END style AMD 7990 LANCE Ethernet network interface driver | 1-190 |
| loadLib | – object module loader | 1-193 |

| | | |
|---------------------|--|-------|
| loginLib | – user login/password subroutine library | 1-194 |
| logLib | – message logging library | 1-195 |
| lptDrv | – parallel chip device driver for the IBM-PC LPT | 1-197 |
| lstLib | – doubly linked list subroutine library | 1-198 |
| m2IcmpLib | – MIB-II ICMP-group API for SNMP Agents | 1-199 |
| m2IfLib | – MIB-II interface-group API for SNMP agents | 1-200 |
| m2IpLib | – MIB-II IP-group API for SNMP agents | 1-201 |
| m2Lib | – MIB-II API library for SNMP agents | 1-203 |
| m2SysLib | – MIB-II system-group API for SNMP agents | 1-206 |
| m2TcpLib | – MIB-II TCP-group API for SNMP agents | 1-207 |
| m2UdpLib | – MIB-II UDP-group API for SNMP agents | 1-209 |
| m68302Sio | – Motorola MC68302 bimodal tty driver | 1-210 |
| m68332Sio | – Motorola MC68332 tty driver | 1-211 |
| m68360Sio | – Motorola MC68360 SCC UART serial driver | 1-211 |
| m68562Sio | – MC68562 DUSCC serial driver | 1-212 |
| m68681Sio | – M68681 serial communications driver | 1-213 |
| m68901Sio | – MC68901 MFP tty driver | 1-215 |
| masterIoLib | – default IO routines for the SNMP master agent | 1-215 |
| mathALib | – C interface library to high-level math functions | 1-218 |
| mathHardLib | – hardware floating-point math library | 1-220 |
| mathSoftLib | – high-level floating-point emulation library | 1-220 |
| mb86940Sio | – MB 86940 UART tty driver | 1-221 |
| mb86960End | – END-style Fujitsu MB86960 Ethernet network interface driver | 1-221 |
| mb87030Lib | – Fujitsu MB87030 SCSI Protocol Controller (SPC) library | 1-223 |
| mbcEnd | – Motorola 68302fads END network interface driver | 1-223 |
| memDrv | – pseudo memory device driver | 1-226 |
| memLib | – full-featured memory partition manager | 1-228 |
| memPartLib | – core memory partition manager | 1-230 |
| memShow | – memory show routines | 1-231 |
| mmanPxLib | – memory management library (POSIX) | 1-231 |
| mmuL64862Lib | – LSI Logic L64862 MBus-to-SBus Interface: I/O DMA library (SPARC) | 1-232 |
| mmuPro32Lib | – mmu library for PentiumPro/II | 1-232 |
| mmuSparcLib | – ROM MMU initialization (SPARC) | 1-237 |
| moduleLib | – object module management library | 1-237 |
| motCpmEnd | – END style Motorola MC68EN360/MPC800 network interface driver | 1-238 |
| motFecEnd | – END style Motorola FEC Ethernet network interface driver | 1-242 |
| mountLib | – Mount protocol library | 1-248 |
| mqPxLib | – message queue library (POSIX) | 1-249 |
| mqPxShow | – POSIX message queue show | 1-250 |
| msgQLib | – message queue library | 1-251 |
| msgQShow | – message queue show routines | 1-252 |
| msgQSmLib | – shared memory message queue library (VxMP Opt.) | 1-252 |
| muxLib | – MUX network interface library | 1-254 |
| ncr710Lib | – NCR 53C710 SCSI I/O Processor (SIOP) library (SCSI-1) | 1-255 |
| ncr710Lib2 | – NCR 53C710 SCSI I/O Processor (SIOP) library (SCSI-2) | 1-256 |

| | | |
|---------------------|---|-------|
| ncr810Lib | – NCR 53C8xx PCI SCSI I/O Processor (SIOP) library (SCSI-2) | 1-257 |
| ncr5390Lib | – NCR5390 SCSI-Bus Interface Controller library (SBIC) | 1-258 |
| ncr5390Lib1 | – NCR 53C90 Advanced SCSI Controller (ASC) library (SCSI-1) | 1-258 |
| ncr5390Lib2 | – NCR 53C90 Advanced SCSI Controller (ASC) library (SCSI-2) | 1-259 |
| ne2000End | – NE2000 END network interface driver | 1-259 |
| nec765Fd | – NEC 765 floppy disk device driver | 1-261 |
| netBufLib | – network buffer library | 1-262 |
| netDrv | – network remote file I/O driver | 1-263 |
| netLib | – network interface library | 1-265 |
| netShow | – network information display routines | 1-265 |
| nfsdLib | – Network File System (NFS) server library | 1-266 |
| nfsDrv | – Network File System (NFS) I/O driver | 1-268 |
| nfsLib | – Network File System (NFS) library | 1-270 |
| nicEvbEnd | – National Semiconductor ST-NIC Chip network interface driver | 1-271 |
| ns16550Sio | – NS 16550 UART tty driver | 1-273 |
| ntEnd | – END network interface driver to ULIP for vxSim for Windows NT | 1-274 |
| ntPassFsLib | – pass-through (to Windows NT) file system library | 1-274 |
| ospfLib | – OSPF version 2 (RFC 1583) routing facilities (OSPF Opt.) | 1-275 |
| passFsLib | – pass-through (to UNIX) file system library (VxSim) | 1-279 |
| pccardLib | – PC CARD enabler library | 1-280 |
| pcic | – Intel 82365SL PCMCIA host bus adaptor chip library | 1-280 |
| pcicShow | – Intel 82365SL PCMCIA host bus adaptor chip show library | 1-281 |
| pcmciaLib | – generic PCMCIA event-handling facilities | 1-281 |
| pcmciaShow | – PCMCIA show library | 1-282 |
| pentiumALib | – Pentium and PentiumPro specific routines | 1-282 |
| pentiumLib | – Pentium and PentiumPro library | 1-286 |
| pentiumShow | – Pentium and PentiumPro specific show routines | 1-288 |
| pingLib | – Packet InterNet Grouper (PING) library | 1-288 |
| pipeDrv | – pipe I/O driver | 1-289 |
| ppc403Sio | – ppc403GA serial driver | 1-291 |
| ppc860Sio | – Motorola MPC800 SMC UART serial driver | 1-292 |
| pppHookLib | – PPP hook library | 1-292 |
| pppLib | – Point-to-Point Protocol library | 1-293 |
| pppSecretLib | – PPP authentication secrets library | 1-295 |
| pppShow | – Point-to-Point Protocol show routines | 1-295 |
| proxyArpLib | – proxy Address Resolution Protocol (ARP) library | 1-296 |
| proxyLib | – proxy Address Resolution Protocol (ARP) client library | 1-297 |
| ptyDrv | – pseudo-terminal driver | 1-297 |
| ramDrv | – RAM disk driver | 1-298 |
| rawFsLib | – raw block device file system library | 1-299 |
| rebootLib | – reboot support library | 1-303 |
| remLib | – remote command library | 1-304 |
| resolvLib | – DNS resolver library | 1-304 |
| ripLib | – Routing Information Protocol (RIP) v1 and v2 library | 1-306 |
| rlogLib | – remote login library | 1-308 |

| | | |
|----------------------|---|-------|
| rngLib | – ring buffer subroutine library | 1-308 |
| routeLib | – network route manipulation library | 1-309 |
| rpcLib | – Remote Procedure Call (RPC) support library | 1-310 |
| rt11FsLib | – RT-11 media-compatible file system library | 1-311 |
| sa1100Sio | – Digital Semiconductor SA-1100 UART tty driver | 1-316 |
| saIoLib | – default transport routines for SNMP subagent | 1-318 |
| schedPxLib | – scheduling library (POSIX) | 1-319 |
| scsi1Lib | – Small Computer System Interface (SCSI) library (SCSI-1) | 1-320 |
| scsi2Lib | – Small Computer System Interface (SCSI) library (SCSI-2) | 1-324 |
| scsiCommonLib | – SCSI library common commands for all devices (SCSI-2) | 1-330 |
| scsiCtrlLib | – SCSI thread-level controller library (SCSI-2) | 1-330 |
| scsiDirectLib | – SCSI library for direct access devices (SCSI-2) | 1-331 |
| scsiLib | – Small Computer System Interface (SCSI) library | 1-332 |
| scsiMgrLib | – SCSI manager library (SCSI-2) | 1-333 |
| scsiSeqLib | – SCSI sequential access device library (SCSI-2) | 1-334 |
| selectLib | – UNIX BSD 4.3 select library | 1-335 |
| semBLib | – binary semaphore library | 1-336 |
| semCLib | – counting semaphore library | 1-338 |
| semLib | – general semaphore library | 1-339 |
| semMLib | – mutual-exclusion semaphore library | 1-341 |
| semOLib | – release 4.x binary semaphore library | 1-344 |
| semPxLib | – semaphore synchronization library (POSIX) | 1-344 |
| semPxShow | – POSIX semaphore show library | 1-346 |
| semShow | – semaphore show routines | 1-346 |
| semSmLib | – shared memory semaphore library (VxMP Opt.) | 1-347 |
| shellLib | – shell execution routines | 1-348 |
| sigLib | – software signal facility library | 1-349 |
| smMemLib | – shared memory management library (VxMP Opt.) | 1-355 |
| smMemShow | – shared memory management show routines (VxMP Opt.) | 1-357 |
| smNameLib | – shared memory objects name database library (VxMP Opt.) | 1-357 |
| smNameShow | – shared memory objects name database show routines (VxMP Opt.) | 1-360 |
| smNetLib | – VxWorks interface to the shared memory network (backplane) driver | 1-360 |
| smNetShow | – shared memory network driver show routines | 1-361 |
| smObjLib | – shared memory objects library (VxMP Opt.) | 1-361 |
| smObjShow | – shared memory objects show routines (VxMP Opt.) | 1-364 |
| sn83932End | – Nat. Semi DP83932B SONIC Ethernet driver | 1-364 |
| sntpcLib | – Simple Network Time Protocol (SNTP) client library | 1-366 |
| sntpsLib | – Simple Network Time Protocol (SNTP) server library | 1-367 |
| sockLib | – generic socket library | 1-368 |
| spyLib | – spy CPU activity library | 1-369 |
| sramDrv | – PCMCIA SRAM device driver | 1-370 |
| st16552Sio | – ST 16C552 DUART tty driver | 1-371 |
| subagentLib | – encode, decode, and process agent and subagent messages | 1-373 |
| symLib | – symbol table subroutine library | 1-374 |
| symSyncLib | – host/target symbol table synchronization | 1-376 |

| | | |
|---------------------|--|-------|
| sysLib | – system-dependent library | 1-378 |
| tapeFsLib | – tape sequential device file system library | 1-380 |
| taskArchLib | – architecture-specific task management routines | 1-384 |
| taskHookLib | – task hook library | 1-384 |
| taskHookShow | – task hook show routines | 1-386 |
| taskInfo | – task information library | 1-386 |
| taskLib | – task management library | 1-387 |
| taskShow | – task show routines | 1-389 |
| taskVarLib | – task variables support library | 1-390 |
| tcic | – Databook TCIC/2 PCMCIA host bus adaptor chip driver | 1-390 |
| tcicShow | – Databook TCIC/2 PCMCIA host bus adaptor chip show library | 1-391 |
| tcpShow | – TCP information display routines | 1-391 |
| telnetLib | – telnet server library | 1-392 |
| tftpdLib | – Trivial File Transfer Protocol server library | 1-392 |
| tftpLib | – Trivial File Transfer Protocol (TFTP) client library | 1-393 |
| tickLib | – clock tick support library | 1-395 |
| timerLib | – timer library (POSIX) | 1-395 |
| timexLib | – execution timer facilities | 1-396 |
| ttyDrv | – provide terminal device access to serial channels | 1-398 |
| tyLib | – tty driver support library | 1-399 |
| udpShow | – UDP information display routines | 1-403 |
| ultraEnd | – SMC Ultra Elite END network interface driver | 1-404 |
| unixDrv | – UNIX-file disk driver (VxSim for Solaris and VxSim for HP) | 1-405 |
| unixSio | – unix serial driver | 1-407 |
| unldLib | – object module unloading library | 1-408 |
| usrAta | – ATA initialization | 1-409 |
| usrConfig | – user-defined system configuration library | 1-410 |
| usrFd | – floppy disk initialization | 1-410 |
| usrIde | – IDE initialization | 1-411 |
| usrLib | – user interface subroutine library | 1-411 |
| usrScsi | – SCSI initialization | 1-413 |
| usrSmObj | – shared memory object initialization | 1-413 |
| vmBaseLib | – base virtual memory support library | 1-413 |
| vmLib | – architecture-independent virtual memory support library (VxVMI Opt.) | 1-414 |
| vmShow | – virtual memory show routines (VxVMI Opt.) | 1-416 |
| vxLib | – miscellaneous support routines | 1-417 |
| VXWList | – simple linked list class (WFC Opt.) | 1-417 |
| VXWMemPart | – memory partition classes (WFC Opt.) | 1-419 |
| VXWModule | – object module class (WFC Opt.) | 1-420 |
| VXWMsgQ | – message queue classes (WFC Opt.) | 1-421 |
| VXWRingBuf | – ring buffer class (WFC Opt.) | 1-422 |
| VXWSem | – semaphore classes (WFC Opt.) | 1-423 |
| VXWSmName | – naming behavior common to all shared memory classes (WFC Opt.) | 1-425 |
| VXWSymTab | – symbol table class (WFC Opt.) | 1-426 |
| VXWTask | – task class (WFC Opt.) | 1-427 |

| | | |
|------------------------|--|-------|
| VXWWd | – watchdog timer class (WFC Opt.) | 1-429 |
| wd33c93Lib | – WD33C93 SCSI-Bus Interface Controller (SBIC) library | 1-430 |
| wd33c93Lib1 | – WD33C93 SCSI-Bus Interface Controller library (SCSI-1) | 1-431 |
| wd33c93Lib2 | – WD33C93 SCSI-Bus Interface Controller library (SCSI-2) | 1-431 |
| wdbEndPktDrv | – END based packet driver for lightweight UDP/IP | 1-432 |
| wdbLib | – WDB agent context management library | 1-432 |
| wdbNetromPktDrv | – NETROM packet driver for the WDB agent | 1-433 |
| wdbPipePktDrv | – pipe packet driver for lightweight UDP/IP | 1-433 |
| wdbSlipPktDrv | – a serial line packetizer for the WDB agent | 1-435 |
| wdbTsfsDrv | – virtual generic file I/O driver for the WDB agent | 1-435 |
| wdbUlIPdrv | – WDB communication interface for the ULIP driver | 1-439 |
| wdbUserEvtLib | – WDB user event library | 1-439 |
| wdbVioDrv | – virtual tty I/O driver for the WDB agent | 1-440 |
| wdLib | – watchdog timer library | 1-440 |
| wdShow | – watchdog show routines | 1-441 |
| winSio | – win serial driver | 1-442 |
| z8530Sio | – Z8530 SCC Serial Communications Controller driver | 1-443 |
| zbufLib | – zbuf interface library | 1-444 |
| zbufSockLib | – zbuf socket interface library | 1-446 |

aic7880Lib

| | |
|--------------------|---|
| NAME | aic7880Lib – Adaptec 7880 SCSI Host Adapter Library File |
| ROUTINES | <i>aic7880CtrlCreate()</i> – create a control structure for the AIC 7880 <i>aic7880ScbCompleted()</i> – successfully completed execution of a client thread <i>aic7880EnableFast20()</i> – enable double speed SCSI data transfers <i>aic7880dFifoThresholdSet()</i> – set the data FIFO threshold. <i>aic7880GetNumOfBuses()</i> – perform a PCI bus scan <i>aic7880ReadConfig()</i> – read from PCI config space <i>aic7880WriteConfig()</i> – read to PCI config space |
| DESCRIPTION | <p>This is the I/O driver for the Adaptec AIC 7880 PCI Bus Master Single Chip SCSI Host Adapter. It is designed to work with scsi2Lib. This driver runs in conjunction with the HIM (Hardware Interface Module) supplied by Adaptec. The AIC 7880 SCSI Host Adapter driver supports the following features:</p> <ul style="list-style-type: none">– Fast, Double Speed 20 MHz data transfers.– 16 bit Wide Synchronous Data transfers.– Tagged Command Queueing.– Data FIFO threshold selection.– Disconnect / Reconnect support.– Multiple Initiator support.– Multiple Controller support. |

In general, the SCSI system and this driver will automatically choose the best combination of these features to suit the target devices used. However, the default choices may be over-ridden by using the function *scsiTargetOptionsSet()* (see **scsiLib**).

OPERATIONS OVERVIEW

The host processor initiates a SCSI I/O operation by programming a data structure called SCB (SCSI Command Block). The SCB contains all the relevant information needed by the Host Adapter to carry out the requested SCSI operation. SCSI SCB's are passed to the HIM by this module which are then sent to the AIC-7880 for execution. The AIC-7880 Sequencer or PhaseEngine comprises the on-chip intelligence that allows the AIC-7880 to execute SCB commands. The Sequencer is programmable and uses its own microcode program which is downloaded to AIC-7880 by the host at initialization.

The following is an example of how an SCB is delivered to the AIC-7880

- Memory is allocated for the SCB structure and it is programmed with the necessary information required to execute a SCSI transaction.
- The SCB is then sent to HIM.
- The HIM pauses the Sequencer.

- The Sequencer has internal registers that point to the area in system memory where the SCB resides.
- The HIM unpauses the Sequencer.
- The AIC-7880 Sequencer uses DMA to transfer the SCB into its internal memory.
- The AIC-7880 executes the SCB.
- Upon completion of the SCB command, the AIC-7880 Sequencer posts the pointer of the completed SCB into system memory.
- The AIC-7880 generates an interrupt.
- The status of the completed SCB is then read by the host.

SCB PROCESSING The AIC-7880 Sequencer uses DMA to transfer the SCB into its internal memory. The Sequencer processes SCB's in the order they are received with new SCB's being started when older SCB operations are idle due to wait for selection or a SCSI bus disconnect. When operations for an Idle SCB reactivate, the sequencer scans the SCB array for the SCB corresponding to the Target/LUN reactivating. The Sequencer then restarts the SCB found until the next disconnect or SCB completion.

MAXIMUM NUMBER OF TAGGED SCB's

The number of tagged SCB's per SCSI target that is handled by the Sequencer, range from 1-32. The HIM supports only the External SCB Access mode. The default number of tags handled by the Sequencer in this mode is 32. Changing the field "Cf_MaxTagScbs" in the cfp_struct changes the maximum number of tagged SCB's.

MAXIMUM NUMBER OF SCB's

The number of SCB's that can be queued to the Sequencer, range from 1-254. This value can be changed before calling the HIM routine "PH_GetConfig ()". Changing the field "Cf_NumberScbs" in "cfp_struct" changes the maximum number of SCB's to be used. The default max number of SCB's is 254.

SYNCHRONOUS TRANSFER SUPPORT

If double speed SCSI mode is enabled, this driver supports transfer periods of 50, 64 and 76 ns. In standard fast SCSI mode transfer periods of 100, 125, 150, 175, 200, 225, 250 and 275 are supported. Synchronous transfer parameters for a target can be set using the SCSI library function "scsiTargetOptionsSet".

DOUBLE SPEED SCSI MODE

To enable/disable double speed SCSI mode the routine "aic7880EnableFast20" needs to be invoked with the following two parameters:

- (1) A pointer to the appropriate SCSI Controller structure
- (2) A BOOLEAN value which enables or disable double speed SCSI mode.

With double speed SCSI mode enabled the host adapter may be capable of transferring data at theoretical transfer rates of 20 MB/s for an 8-bit device and 40 MB/s for a 16-bit device. Double Speed SCSI is disabled by default.

DATA FIFO THRESHOLD

To set the data FIFO threshold the routine "aic7880dFifoThresholdSet" needs to be invoked with the following two parameters:

- (1) A pointer to the appropriate SCSI Controller structure
- (2) The data FIFO threshold value.

For more information about the data FIFO threshold value refer the *aic7880dFifoThresholdSet()* routine

In order to initialize the driver from the BSP the following needs to be done in the BSP specific routine *sysScsiInit()* in file *sysScsi.c*.

- Find the SCSI Host Adapter.
- Create the SCSI Controller Structure.
- Connect the interrupt to Interrupt Service Routine (ISR).
- Enable the SCSI interrupt

The following example shows the SCSI initialization sequence that need to be done in the BSP.

```
STATUS sysScsiInit ()
{
    int    busNo;          /* PCI bus number      */
    int    devNo;         /* PCI device number   */
    UWORD  found = FALSE; /* host adapter found  */
    int    numHa = 0;     /* number of host adapters */
    for (busNo=0; busNo < MAX_NO_OF_PCI_BUSES && !found; busNo++)
        for (devNo = 0; devNo < MAX_NO_OF_PCI_DEVICES; devNo++)
            {
                if ((found = sysScsiHostAdapterFind (busNo, devNo)) == HA_FOUND)
                    {
                        numHa++;
                        /* Create the SCSI controller */
                        if ((pSysScsiCtrl = (SCSI_CTRL *) aic7880CtrlCreate
                            (busNo, devNo, SCSI_DEF_CTRL_BUS_ID)) == NULL)
                            {
                                logMsg ("Could not create SCSI controller\n",
                                        0, 0, 0, 0, 0, 0);
                                return (ERROR);
                            }
                    }
                /* connect the SCSI controller's interrupt service routine */
                if ((intConnect (INUM_TO_IVEC (SCSI_INT_VEC), aic7880Intr,
                                (int) pSysScsiCtrl)) == ERROR)
                    {

```

```
        return (ERROR);  
    /* enable SCSI interrupts */  
    sysIntEnablePIC (SCSI_INT_LVL);  
    }  
    return (OK);  
    }
```

SEE ALSO *scsiLib*, *scsi2Lib*, *cacheLib*, *AIC-7880 Design In Handbook*, *AIC-7880 Data Book*, *Adaptec Hardware Interface Module (HIM) Specification*, *VxWorks Programmer's Guide: I/O System*

aioPxLib

NAME *aioPxLib* – asynchronous I/O (AIO) library (POSIX)

ROUTINES *aioPxLibInit()* – initialize the asynchronous I/O (AIO) library
aio_read() – initiate an asynchronous read (POSIX)
aio_write() – initiate an asynchronous write (POSIX)
lio_listio() – initiate a list of asynchronous I/O requests (POSIX)
aio_suspend() – wait for asynchronous I/O request(s) (POSIX)
aio_fsync() – asynchronous file synchronization (POSIX)
aio_error() – retrieve error status of asynchronous I/O operation (POSIX)
aio_return() – retrieve return status of asynchronous I/O operation (POSIX)

DESCRIPTION This library implements asynchronous I/O (AIO) according to the definition given by the POSIX standard 1003.1b (formerly 1003.4, Draft 14). AIO provides the ability to overlap application processing and I/O operations initiated by the application. With AIO, a task can perform I/O simultaneously to a single file multiple times or to multiple files.

After an AIO operation has been initiated, the AIO proceeds in logical parallel with the processing done by the application. The effect of issuing an asynchronous I/O request is as if a separate thread of execution were performing the requested I/O.

AIO LIBRARY The AIO library is initialized by calling *aioPxLibInit()*, which should be called once (typically at system start-up) after the I/O system has already been initialized.

AIO COMMANDS The file to be accessed asynchronously is opened via the standard open call. Open returns a file descriptor which is used in subsequent AIO calls.

The caller initiates asynchronous I/O via one of the following routines:

aio_read()
initiates an asynchronous read

aiowrite()
initiates an asynchronous write

lio_listio()
initiates a list of asynchronous I/O requests

Each of these routines has a return value and error value associated with it; however, these values indicate only whether the AIO request was successfully submitted (queued), not the ultimate success or failure of the AIO operation itself.

There are separate return and error values associated with the success or failure of the AIO operation itself. The error status can be retrieved using *aiowerror()*; however, until the AIO operation completes, the error status will be `EINPROGRESS`. After the AIO operation completes, the return status can be retrieved with *aioreturn()*.

The *aiocancel()* call cancels a previously submitted AIO request. The *aiosuspend()* call waits for an AIO operation to complete.

Finally, the *aioshow()* call (not a standard POSIX function) displays outstanding AIO requests.

AIO CONTROL BLOCK

Each of the calls described above takes an AIO control block (**aiocb**) as an argument. The calling routine must allocate space for the **aiocb**, and this space must remain available for the duration of the AIO operation. (Thus the **aiocb** must not be created on the task's stack unless the calling routine will not return until after the AIO operation is complete and *aioreturn()* has been called.) Each **aiocb** describes a single AIO operation. Therefore, simultaneous asynchronous I/O operations using the same **aiocb** are not valid and produce undefined results.

The **aiocb** structure and the data buffers referenced by it are used by the system to perform the AIO request. Therefore, once the **aiocb** has been submitted to the system, the application must not modify the **aiocb** structure until after a subsequent call to *aioreturn()*. The *aioreturn()* call retrieves the previously submitted AIO data structures from the system. After the *aioreturn()* call, the calling application can modify the **aiocb**, free the memory it occupies, or reuse it for another AIO call.

As a result, if space for the **aiocb** is allocated off the stack the task should not be deleted (or complete running) until the **aiocb** has been retrieved from the system via an *aioreturn()*.

The **aiocb** is defined in **aiio.h**. It has the following elements:

```
struct
{
    int                aio_fildes;
    off_t              aio_offset;
    volatile void *    aio_buf;
    size_t             aio_nbytes;
    int                aio_reqprio;
```

```
struct sigevent      aio_sigevent;  
int                  aio_lio_opcode;  
AIO_SYS              aio_sys;  
} aiocb
```

aio_fildes

file descriptor for I/O.

aio_offset

offset from the beginning of the file where the AIO takes place. Note that performing AIO on the file does not cause the offset location to automatically increase as in read and write; the caller must therefore keep track of the location of reads and writes made to the file (see POSIX COMPLIANCE below).

aio_buf

address of the buffer from/to which AIO is requested.

aio_nbytes

number of bytes to read or write.

aio_reqprio

amount by which to lower the priority of an AIO request. Each AIO request is assigned a priority; this priority, based on the calling task's priority, indicates the desired order of execution relative to other AIO requests for the file. The **aio_reqprio** member allows the caller to lower (but not raise) the AIO operation priority by the specified value. Valid values for **aio_reqprio** are in the range of zero through **AIO_PRIO_DELTA_MAX**. If the value specified by **aio_reqprio** results in a priority lower than the lowest possible task priority, the lowest valid task priority is used.

aio_sigevent

(optional) if nonzero, the signal to return on completion of an operation.

aio_lio_opcode

operation to be performed by a *lio_listio()* call; valid entries include **LIO_READ**, **LIO_WRITE**, and **LIO_NOP**.

aio_sys

a Wind River Systems addition to the **aiocb** structure; it is used internally by the system and must not be modified by the user.

EXAMPLES

A writer could be implemented as follows:

```
if ((pAioWrite = calloc (1, sizeof (struct aiocb))) == NULL)  
{  
    printf ("calloc failed\n");  
    return (ERROR);  
}  
pAioWrite->aio_fildes = fd;  
pAioWrite->aio_buf = buffer;  
pAioWrite->aio_offset = 0;
```

```
strcpy (pAioWrite->aio_buf, "test string");
pAioWrite->aio_nbytes = strlen ("test string");
pAioWrite->aio_sigevent.sigev_notify = SIGEV_NONE;
aio_write (pAioWrite);
/* .
.
do other work
.
.
*/
/* now wait until I/O finishes */
while (aio_error (pAioWrite) == EINPROGRESS)
    taskDelay (1);
aio_return (pAioWrite);
free (pAioWrite);
```

A reader could be implemented as follows:

```
/* initialize signal handler */

action1.sa_sigaction = sigHandler;
action1.sa_flags = SA_SIGINFO;
sigemptyset(&action1.sa_mask);
sigaction (TEST_RT_SIG1, &action1, NULL);
if ((pAioRead = calloc (1, sizeof (struct aiocb))) == NULL)
{
    printf ("calloc failed\n");
    return (ERROR);
}

pAioRead->aio_fildes = fd;
pAioRead->aio_buf = buffer;
pAioRead->aio_nbytes = BUF_SIZE;
pAioRead->aio_sigevent.sigev_signo = TEST_RT_SIG1;
pAioRead->aio_sigevent.sigev_notify = SIGEV_SIGNAL;
pAioRead->aio_sigevent.sigev_value.sival_ptr = (void *)pAioRead;

aio_read (pAioRead);
/*
.
.
do other work
.
.
*/
```

The signal handler might look like the following:

```
void sigHandler
(
    int          sig,
    struct siginfo info,
    void *       pContext
)
{
    struct aiocb * pAioDone;
    pAioDone = (struct aiocb *) info.si_value.sival_ptr;
    aio_return (pAioDone);
    free (pAioDone);
}
```

POSIX COMPLIANCE

Currently VxWorks does not support the `O_APPEND` flag in the open call. Therefore, the user must keep track of the offset in the file that the asynchronous writes occur (as in the case of reads). The `aio_offset` field is used to specify that file position.

In addition, VxWorks does not currently support synchronized I/O.

INCLUDE FILES `aio.h`

SEE ALSO POSIX 1003.1b document

aioPxShow

NAME `aioPxShow` – asynchronous I/O (AIO) show library

ROUTINES `aioShow()` – show AIO requests

DESCRIPTION This library implements the show routine for **`aioPxLib`**.

aioSysDrv

| | |
|--------------------|--|
| NAME | aioSysDrv – AIO system driver |
| ROUTINES | <i>aioSysInit()</i> – initialize the AIO system driver |
| DESCRIPTION | This library is the AIO system driver. The system driver implements asynchronous I/O with system AIO tasks performing the AIO requests in a synchronous manner. It is installed as the default driver for AIO. |
| SEE ALSO | POSIX 1003.1b document |

ambaSio

| | |
|--------------------|---|
| NAME | ambaSio – ARM AMBA UART tty driver |
| ROUTINES | <i>ambaDevInit()</i> – initialise an AMBA channel <i>ambaIntTx()</i> – handle a transmitter interrupt <i>ambaIntRx()</i> – handle a receiver interrupt |
| DESCRIPTION | <p>This is the device driver for the Advanced RISC Machines (ARM) AMBA UART. This is a generic design of UART used within a number of chips containing (or for use with) ARM CPUs such as in the Digital Semiconductor 21285 chip as used in the EBSA-285 BSP.</p> <p>This design contains a universal asynchronous receiver/transmitter, a baud-rate generator, and an InfraRed Data Association (IrDa) Serial InfraRed (SiR) protocol encoder. The SiR encoder is not supported by this driver. The UART contains two 16-entry deep FIFOs for receive and transmit: if a framing, overrun or parity error occurs during reception, the appropriate error bits are stored in the receive FIFO along with the received data. The FIFOs can be programmed to be one byte deep only, like a conventional UART with double buffering, but the only mode of operation supported is with the FIFOs enabled.</p> <p>The UART design does not support the modem control output signals: DTR, RI and RTS. Moreover, the implementation in the 21285 chip does not support the modem control inputs: DCD, CTS and DSR.</p> <p>The UART design can generate four interrupts: Rx, Tx, modem status change and a UART disabled interrupt (which is asserted when a start bit is detected on the receive line when the UART is disabled). The implementation in the 21285 chip has only two interrupts: Rx and Tx, but the Rx interrupt is a combination of the normal Rx interrupt status and the UART disabled interrupt status.</p> |

Only asynchronous serial operation is supported by the UART which supports 5 to 8 bit word lengths with or without parity and with one or two stop bits. The only serial word format supported by the driver is 8 data bits, 1 stop bit, no parity. The default baud rate is determined by the BSP by filling in the `AMBA_CHAN` structure before calling `ambaDevInit()`.

The exact baud rates supported by this driver will depend on the crystal fitted (and consequently the input clock to the baud-rate generator), but in general, baud rates from about 300 to about 115200 are possible.

In theory, any number of UART channels could be implemented within a chip. This driver has been designed to cope with an arbitrary number of channels, but at the time of writing, has only ever been tested with one channel.

DATA STRUCTURES

An `AMBA_CHAN` data structure is used to describe each channel, this structure is described in `h/drv/sio/ambaSio.h`.

CALLBACKS

Servicing a "transmitter ready" interrupt involves making a callback to a higher level library in order to get a character to transmit. By default, this driver installs dummy callback routines which do nothing. A higher layer library that wants to use this driver (e.g. `ttyDrv`) will install its own callback routine using the `SIO_INSTALL_CALLBACK` ioctl command. Likewise, a receiver interrupt handler makes a callback to pass the character to the higher layer library.

MODES

This driver supports both polled and interrupt modes.

USAGE

The driver is typically only called by the BSP. The directly callable routines in this module are `ambaDevInit()`, `ambaIntTx()` and `ambaIntRx()`.

The BSP's `sysHwInit()` routine typically calls `sysSerialHwInit()`, which initialises the hardware-specific fields in the `AMBA_CHAN` structure (e.g. register I/O addresses etc) before calling `ambaDevInit()` which resets the device and installs the driver function pointers. After this the UART will be enabled and ready to generate interrupts, but those interrupts will be disabled in the interrupt controller.

The following example shows the first parts of the initialisation:

```
#include "drv/sio/ambaSio.h"
LOCAL AMBA_CHAN ambaChan[N_AMBA_UART_CHANS];
void sysSerialHwInit (void)
{
    int i;
    for (i = 0; i < N_AMBA_UART_CHANS; i++)
    {
        ambaChan[i].regs = devParas[i].baseAdrs;
        ambaChan[i].baudRate = CONSOLE_BAUD_RATE;
        ambaChan[i].xtal = UART_XTAL_FREQ;
    }
}
```

```

        ambaChan[i].levelRx = devParas[i].intLevelRx;
        ambaChan[i].levelTx = devParas[i].intLevelTx;
    /*
     * Initialise driver functions, getTxChar, putRcvChar and
     * channelMode, then initialise UART
     */
    ambaDevInit(&ambaChan[i]);
    }
}

```

The BSP's *sysHwInit2()* routine typically calls *sysSerialHwInit2()*, which connects the chips interrupts via *intConnect()* (the two interrupts **ambaIntTx** and **ambaIntRx**) and enables those interrupts, as shown in the following example:

```

void sysSerialHwInit2 (void)
{
    /* connect and enable Rx interrupt */
    (void) intConnect (INUM_TO_IVEC(devParas[0].vectorRx),
                     ambaIntRx, (int) &ambaChan[0]);
    intEnable (devParas[0].intLevelRx);
    /* connect Tx interrupt */
    (void) intConnect (INUM_TO_IVEC(devParas[0].vectorTx),
                     ambaIntTx, (int) &ambaChan[0]);
    /*
     * There is no point in enabling the Tx interrupt, as it will
     * interrupt immediately and then be disabled.
     */
}

```

BSP

By convention all the BSP-specific serial initialisation is performed in a file called **sysSerial.c**, which is #include'd by **sysLib.c**. **sysSerial.c** implements at least four functions, *sysSerialHwInit()*, *sysSerialHwInit2()*, *sysSerialChanGet()*, and *sysSerialReset()*. The first two have been described above, the others work as follows:

sysSerialChanGet() is called by *usrRoot* to get the serial channel descriptor associated with a serial channel number. The routine takes a single parameter which is a channel number ranging between zero and **NUM_TTY**. It returns a pointer to the corresponding channel descriptor, **SIO_CHAN ***, which is just the address of the **AMBA_CHAN** structure.

sysSerialReset() is called from *sysToMonitor()* and should reset the serial devices to an inactive state (prevent them from generating any interrupts).

INCLUDE FILES

drv/sio/ambaSio.h **sioLib.h**

SEE ALSO

Advanced RISC Machines AMBA UART (AP13) Data Sheet, Digital Semiconductor 21285 Core Logic for SA-110 Microprocessor Data Sheet, " Digital Semiconductor EBSA-285 Evaluation Board Reference Manual.

ansiAssert

| | |
|----------------------|---|
| NAME | ansiAssert – ANSI assert documentation |
| ROUTINES | <i>assert()</i> – put diagnostics into programs (ANSI) |
| DESCRIPTION | The header assert.h defines the <i>assert()</i> macro and refers to another macro, NDEBUG , which is not defined by assert.h . If NDEBUG is defined as a macro at the point in the source file where assert.h is included, the <i>assert()</i> macro is defined simply as: <pre>#define assert(ignore) ((void)0)</pre> ANSI specifies that <i>assert()</i> should be implemented as a macro, not as a routine. If the macro definition is suppressed in order to access an actual routine, the behavior is undefined. |
| INCLUDE FILES | stdio.h , stdlib.h , assert.h |
| SEE ALSO | American National Standard X3.159-1989 |

ansiCtype

| | |
|--------------------|---|
| NAME | ansiCtype – ANSI ctype documentation |
| ROUTINES | <i>isalnum()</i> – test whether a character is alphanumeric (ANSI) <i>isalpha()</i> – test whether a character is a letter (ANSI) <i>iscntrl()</i> – test whether a character is a control character (ANSI) <i>isdigit()</i> – test whether a character is a decimal digit (ANSI) <i>isgraph()</i> – test whether a character is a printing, non-white-space character (ANSI) <i>islower()</i> – test whether a character is a lower-case letter (ANSI) <i>isprint()</i> – test whether a character is printable, including the space character (ANSI) <i>ispunct()</i> – test whether a character is punctuation (ANSI) <i>isspace()</i> – test whether a character is a white-space character (ANSI) <i>isupper()</i> – test whether a character is an upper-case letter (ANSI) <i>isxdigit()</i> – test whether a character is a hexadecimal digit (ANSI) <i>tolower()</i> – convert an upper-case letter to its lower-case equivalent (ANSI) <i>toupper()</i> – convert a lower-case letter to its upper-case equivalent (ANSI) |
| DESCRIPTION | The header ctype.h declares several functions useful for testing and mapping characters. In all cases, the argument is an int , the value of which is representable as an unsigned |

char or is equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior of the **ctype** functions is affected by the current locale. VxWorks supports only the "C" locale.

The term "printing character" refers to a member of an implementation-defined set of characters, each of which occupies one printing position on a display device; the term "control character" refers to a member of an implementation-defined set of characters that are not printing characters.

| | |
|----------------------|--|
| INCLUDE FILES | ctype.h |
| SEE ALSO | American National Standard X3.159-1989 |

ansiLocale

| | |
|--------------------|---|
| NAME | ansiLocale – ANSI locale documentation |
| ROUTINES | <i>localeconv()</i> – set the components of an object with type lconv (ANSI) <i>setlocale()</i> – set the appropriate locale (ANSI) |
| DESCRIPTION | The header locale.h declares two functions and one type, and defines several macros. The type is: struct lconv contains members related to the formatting of numeric values. The structure should contain at least the members defined in locale.h , in any order. |
| SEE ALSO | <i>localeconv()</i> , <i>setlocale()</i> , American National Standard X3.159-1989 |

ansiMath

| | |
|-----------------|---|
| NAME | ansiMath – ANSI math documentation |
| ROUTINES | <i>asin()</i> – compute an arc sine (ANSI) <i>acos()</i> – compute an arc cosine (ANSI) <i>atan()</i> – compute an arc tangent (ANSI) <i>atan2()</i> – compute the arc tangent of y/x (ANSI) <i>ceil()</i> – compute the smallest integer greater than or equal to a specified value (ANSI) |

ansiMath

cosh() – compute a hyperbolic cosine (ANSI)
exp() – compute an exponential value (ANSI)
fabs() – compute an absolute value (ANSI)
floor() – compute the largest integer less than or equal to a specified value (ANSI)
fmod() – compute the remainder of x/y (ANSI)
frexp() – break a floating-point number into a normalized fraction and power of 2 (ANSI)
ldexp() – multiply a number by an integral power of 2 (ANSI)
log() – compute a natural logarithm (ANSI)
log10() – compute a base-10 logarithm (ANSI)
modf() – separate a floating-point number into integer and fraction parts (ANSI)
pow() – compute the value of a number raised to a specified power (ANSI)
sin() – compute a sine (ANSI)
cos() – compute a cosine (ANSI)
sinh() – compute a hyperbolic sine (ANSI)
sqrt() – compute a non-negative square root (ANSI)
tan() – compute a tangent (ANSI)
tanh() – compute a hyperbolic tangent (ANSI)

DESCRIPTION The header **math.h** declares several mathematical functions and defines one macro. The functions take double arguments and return double values.

The macro defined is:

HUGE_VAL

expands to a positive double expression, not necessarily representable as a float.

The behavior of each of these functions is defined for all representable values of their input arguments. Each function executes as if it were a single operation, without generating any externally visible exceptions.

For all functions, a domain error occurs if an input argument is outside the domain over which the mathematical function is defined. The description of each function lists any applicable domain errors. On a domain error, the function returns an implementation-defined value; the value EDOM is stored in **errno**.

Similarly, a range error occurs if the result of the function cannot be represented as a double value. If the result overflows (the magnitude of the result is so large that it cannot be represented in an object of the specified type), the function returns the value **HUGE_VAL**, with the same sign (except for the **tan()** function) as the correct value of the function; the value ERANGE is stored in **errno**. If the result underflows (the type), the function returns zero; whether the integer expression **errno** acquires the value ERANGE is implementation defined.

INCLUDE FILES **math.h**

SEE ALSO **mathALib**, American National Standard X3.159-1989

ansiSetjmp

| | |
|--------------------|---|
| NAME | ansiSetjmp – ANSI setjmp documentation |
| ROUTINES | <i>setjmp()</i> – save the calling environment in a jmp_buf argument (ANSI) <i>longjmp()</i> – perform non-local goto by restoring saved environment (ANSI) |
| DESCRIPTION | The header setjmp.h defines functions and one type for bypassing the normal function call and return discipline. The type declared is: jmp_buf an array type suitable for holding the information needed to restore a calling environment. The ANSI C standard does not specify whether <i>setjmp()</i> is a subroutine or a macro. |
| SEE ALSO | American National Standard X3.159-1989 |

ansiStdarg

| | |
|--------------------|--|
| NAME | ansiStdarg – ANSI stdarg documentation |
| ROUTINES | <i>va_start()</i> – initialize a va_list object for use by <i>va_arg()</i> and <i>va_end()</i> <i>va_arg()</i> – expand to an expression having the type and value of the call's next argument <i>va_end()</i> – facilitate a normal return from a routine using a va_list object |
| DESCRIPTION | The header stdarg.h declares a type and defines three macros for advancing through a list of arguments whose number and types are not known to the called function when it is translated. A function may be called with a variable number of arguments of varying types. The rightmost parameter plays a special role in the access mechanism, and is designated <i>parmN</i> in this description. The type declared is: va_list a type suitable for holding information needed by the macros <i>va_start()</i> , <i>va_arg()</i> , and <i>va_end()</i> . To access the varying arguments, the called function shall declare an object having type va_list . The object (referred to here as <i>ap</i>) may be passed as an argument to another |

function; if that function invokes the *va_arg()* macro with parameter *ap*, the value of *ap* in the calling function is indeterminate and is passed to the *va_end()* macro prior to any further reference to *ap*.

va_start() and *va_arg()* have been implemented as macros, not as functions. The *va_start()* and *va_end()* macros should be invoked in the function accepting a varying number of arguments, if access to the varying arguments is desired.

The use of these macros is documented here as if they were architecture-generic. However, depending on the compilation environment, different macro versions are included by **vxWorks.h**.

SEE ALSO

American National Standard X3.159-1989

ansiStdio

NAME

ansiStdio – ANSI **stdio** documentation

ROUTINES

clearerr() – clear end-of-file and error flags for a stream (ANSI)
fclose() – close a stream (ANSI)
fdopen() – open a file specified by a file descriptor (POSIX)
feof() – test the end-of-file indicator for a stream (ANSI)
ferror() – test the error indicator for a file pointer (ANSI)
fflush() – flush a stream (ANSI)
fgetc() – return the next character from a stream (ANSI)
fgetpos() – store the current value of the file position indicator for a stream (ANSI)
fgets() – read a specified number of characters from a stream (ANSI)
fileno() – return the file descriptor for a stream (POSIX)
fopen() – open a file specified by name (ANSI)
fprintf() – write a formatted string to a stream (ANSI)
fputc() – write a character to a stream (ANSI)
fputs() – write a string to a stream (ANSI)
fread() – read data into an array (ANSI)
freopen() – open a file specified by name (ANSI)
fscanf() – read and convert characters from a stream (ANSI)
fseek() – set the file position indicator for a stream (ANSI)
fsetpos() – set the file position indicator for a stream (ANSI)
ftell() – return the current value of the file position indicator for a stream (ANSI)
fwrite() – write from a specified array (ANSI)
getc() – return the next character from a stream (ANSI)
getchar() – return the next character from the standard input stream (ANSI)
gets() – read characters from the standard input stream (ANSI)
getw() – read the next word (32-bit integer) from a stream

perror() – map an error number in **errno** to an error message (ANSI)
 putchar() – write a character to a stream (ANSI)
 putchar() – write a character to the standard output stream (ANSI)
 puts() – write a string to the standard output stream (ANSI)
 putw() – write a word (32-bit integer) to a stream
 rewind() – set the file position indicator to the beginning of a file (ANSI)
 scanf() – read and convert characters from the standard input stream (ANSI)
 setbuf() – specify the buffering for a stream (ANSI)
 setbuffer() – specify buffering for a stream
 setlinebuf() – set line buffering for standard output or standard error
 setvbuf() – specify buffering for a stream (ANSI)
 stdioInit() – initialize standard I/O support
 stdioFp() – return the standard input/output/error FILE of the current task
 stdioShowInit() – initialize the standard I/O show facility
 stdioShow() – display file pointer internals
 tmpfile() – create a temporary binary file (Unimplemented) (ANSI)
 tmpnam() – generate a temporary file name (ANSI)
 ungetc() – push a character back into an input stream (ANSI)
 vfprintf() – write a formatted string to a stream (ANSI)

DESCRIPTION The header **stdio.h** declares three types, several macros, and many functions for performing input and output.

Types The types declared are **size_t** and:

FILE

object type capable of recording all the information needed to control a stream, including its file position indicator, a pointer to its associated buffer (if any), an error indicator that records whether a read/write error has occurred, and an end-of-file indicator that records whether the end of the file has been reached.

fpos_t

object type capable of recording all the information needed to specify uniquely every position within a file.

Macros The macros are NULL and:

_IOFBF, _IOLBF, _IONBF

expand to integral constant expressions with distinct values, suitable for use as the third argument to *setvbuf()*.

BUFSIZ

expands to an integral constant expression that is the size of the buffer used by *setbuf()*.

EOF

expands to a negative integral constant expression that is returned by several functions to indicate **end-of-file**, that is, no more input from a stream.

FOPEN_MAX

expands to an integral constant expression that is the minimum number of the files that the system guarantees can be open simultaneously.

FILENAME_MAX

expands to an integral constant expression that is the size needed for an array of **char** large enough to hold the longest file name string that can be used.

L_tmpnam

expands to an integral constant expression that is the size needed for an array of **char** large enough to hold a temporary file name string generated by *tmpnam()*.

SEEK_CUR, SEEK_END, SEEK_SET

expand to integral constant expressions with distinct values suitable for use as the third argument to *fseek()*.

TMP_MAX

expands to an integral constant expression that is the minimum number of file names generated by *tmpnam()* that will be unique.

stderr, stdin, stdout

expressions of type "pointer to FILE" that point to the FILE objects associated, respectively, with the standard error, input, and output streams.

STREAMS

Input and output, whether to or from physical devices such as terminals and tape drives, or whether to or from files supported on structured storage devices, are mapped into logical data streams, whose properties are more uniform than their various inputs and outputs. Two forms of mapping are supported: for text streams and for binary streams.

A text stream is an ordered sequence of characters composed into lines, each line consisting of zero or more characters plus a terminating new-line character. Characters may have to be added, altered, or deleted on input and output to conform to differing conventions for representing text in the host environment. Thus, there is no need for a one-to-one correspondence between the characters in a stream and those in the external representation. Data read in from a text stream will necessarily compare equal to the data that were earlier written out to that stream only if: the data consists only of printable characters and the control characters horizontal tab and new-line; no new-line character is immediately preceded by space characters; and the last character is a new-line character. Space characters are written out immediately before a new-line character appears.

A binary stream is an ordered sequence of characters that can transparently record internal data. Data read in from a binary stream should compare equal to the data that was earlier written out to that stream, under the same implementation. However, such a stream may have a number of null characters appended to the end of the stream.

Environmental Limits

VxWorks supports text files with lines containing at least 254 characters, including the terminating new-line character. The value of the macro BUFSIZ is 1024.

FILES

A stream is associated with an external file (which may be a physical device) by opening a file, which may involve creating a new file. Creating an existing file causes its former contents to be discarded, if necessary. If a file can support positioning requests (such as a disk file, as opposed to a terminal), then a file position indicator associated with the stream is positioned at the start (character number zero) of the file. The file position indicator is maintained by subsequent reads, writes, and positioning requests, to facilitate an orderly progression through the file. All input takes place as if characters were read by successive calls to *fgetc()*; all output takes place as if characters were written by successive calls to *fputc()*.

Binary files are not truncated, except as defined in *fopen()* documentation.

When a stream is unbuffered, characters are intended to appear from the source or at the destination as soon as possible. Otherwise characters may be accumulated and transmitted to or from the host environment as a block. When a stream is fully buffered, characters are intended to be transmitted to or from the host environment as a block when the buffer is filled. When a stream is line buffered, characters are intended to be transmitted to or from the host environment as a block when a new-line character is encountered. Furthermore, characters are intended to be transmitted as a block to the host environment when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line-buffered stream that requires the transmission of characters from the host environment. VxWorks supports these characteristics via the *setbuf()* and *setvbuf()* functions.

A file may be disassociated from a controlling stream by closing the file. Output streams are flushed (any unwritten buffer contents are transmitted to the host environment) before the stream is disassociated from the file. The value of a pointer to a FILE object is indeterminate after the associated file is closed (including the standard text streams).

The file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at its start).

TASK TERMINATION

ANSI specifies that if the main function returns to its original caller or if *exit()* is called, all open files are closed (and hence all output streams are flushed) before program termination. This does **not** happen in VxWorks. The *exit()* function does not close all files opened for that task. A file opened by one task may be used and closed by another. Unlike in UNIX, when a VxWorks task exits, it is the responsibility of the task to *fclose()* its file pointers, except *stdin*, *stdout*, and *stderr*. If a task is to be terminated asynchronously, use *kill()* and arrange for a signal handler to clean up.

The address of the FILE object used to control a stream may be significant; a copy of a FILE object may not necessarily serve in place of the original.

At program startup, three text streams are predefined and need not be opened explicitly: standard input (for reading conventional input), standard output (for writing conventional output), and standard error (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output

streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

Functions that open additional (non-temporary) files require a file name, which is a string. VxWorks allows the same file to be open multiple times simultaneously. It is up to the user to maintain synchronization between different tasks accessing the same file.

FIOLIB Several routines normally considered part of standard I/O -- *printf()*, *sprintf()*, *vprintf()*, *vsprintf()*, and *scanf()* -- are not implemented as part of the buffered standard I/O library; they are instead implemented in **fiolib**. They do not use the standard I/O buffering scheme. They are self-contained, formatted, but unbuffered I/O functions. This allows a limited amount of formatted I/O to be achieved without the overhead of the standard I/O library.

SEE ALSO **fiolib**, *American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: Input/Output (stdio.h)*

ansiStdlib

NAME **ansiStdlib** – ANSI **stdlib** documentation

ROUTINES

- abort()* – cause abnormal program termination (ANSI)
- abs()* – compute the absolute value of an integer (ANSI)
- atexit()* – call a function at program termination (Unimplemented) (ANSI)
- atof()* – convert a string to a **double** (ANSI)
- atoi()* – convert a string to an **int** (ANSI)
- atol()* – convert a string to a **long** (ANSI)
- bsearch()* – perform a binary search (ANSI)
- div()* – compute a quotient and remainder (ANSI)
- div_r()* – compute a quotient and remainder (reentrant)
- labs()* – compute the absolute value of a **long** (ANSI)
- ldiv()* – compute the quotient and remainder of the division (ANSI)
- ldiv_r()* – compute a quotient and remainder (reentrant)
- mblen()* – calculate the length of a multibyte character (Unimplemented) (ANSI)
- mbtowc()* – convert a multibyte character to a wide character (Unimplemented) (ANSI)
- wctomb()* – convert a wide character to a multibyte character (Unimplemented) (ANSI)
- mbstowcs()* – convert a series of multibyte char's to wide char's (Unimplemented) (ANSI)
- wcstombs()* – convert a series of wide char's to multibyte char's (Unimplemented) (ANSI)
- qsort()* – sort an array of objects (ANSI)
- rand()* – generate a pseudo-random integer between 0 and **RAND_MAX** (ANSI)
- srand()* – reset the value of the seed used to generate random numbers (ANSI)
- strtod()* – convert the initial portion of a string to a double (ANSI)
- strtol()* – convert a string to a long integer (ANSI)

strtoul() – convert a string to an unsigned long integer (ANSI)
system() – pass a string to a command processor (Unimplemented) (ANSI)

DESCRIPTION This library includes several standard ANSI routines. Note that where there is a pair of routines, such as *div()* and *div_r()*, only the routine *xxx_r()* is reentrant. The *xxx()* routine is not reentrant.

The header **stdlib.h** declares four types and several functions of general utility, and defines several macros.

Types The types declared are **size_t**, **wchar_t**, and:

div_t
is the structure type of the value returned by the *div()*.

ldiv_t
is the structure type of the value returned by the *ldiv_t()*.

Macros The macros defined are NULL and:

EXIT_FAILURE, **EXIT_SUCCESS**
expand to integral constant expressions that may be used as the argument to *exit()* to return unsuccessful or successful termination status, respectively, to the host environment.

RAND_MAX
expands to a positive integer expression whose value is the maximum number of bytes on a multibyte character for the extended character set specified by the current locale, and whose value is never greater than **MB_LEN_MAX**.

INCLUDE FILES **stdlib.h**

SEE ALSO American National Standard X3.159-1989

ansiString

NAME **ansiString** – ANSI **string** documentation

ROUTINES *memchr()* – search a block of memory for a character (ANSI)
memcmp() – compare two blocks of memory (ANSI)
memcpy() – copy memory from one location to another (ANSI)
memmove() – copy memory from one location to another (ANSI)
memset() – set a block of memory (ANSI)
strcat() – concatenate one string to another (ANSI)
strchr() – find the first occurrence of a character in a string (ANSI)

ansiTime

strcmp() – compare two strings lexicographically (ANSI)
strcoll() – compare two strings as appropriate to LC_COLLATE (ANSI)
strcpy() – copy one string to another (ANSI)
strcspn() – return the string length up to the first character from a given set (ANSI)
strerror_r() – map an error number to an error string (POSIX)
strerror() – map an error number to an error string (ANSI)
strlen() – determine the length of a string (ANSI)
strncat() – concatenate characters from one string to another (ANSI)
strncmp() – compare the first *n* characters of two strings (ANSI)
strncpy() – copy characters from one string to another (ANSI)
strpbrk() – find the first occurrence in a string of a character from a given set (ANSI)
strrchr() – find the last occurrence of a character in a string (ANSI)
strspn() – return the string length up to the first character not in a given set (ANSI)
strstr() – find the first occurrence of a substring in a string (ANSI)
strtok() – break down a string into tokens (ANSI)
strtok_r() – break down a string into tokens (reentrant) (POSIX)
strxfrm() – transform up to *n* characters of *s2* into *s1* (ANSI)

DESCRIPTION

This library includes several standard ANSI routines. Note that where there is a pair of routines, such as *div()* and *div_r()*, only the routine *xxx_r()* is reentrant. The *xxx()* routine is not reentrant.

The header **string.h** declares one type and several functions, and defines one macro useful for manipulating arrays of character type and other objects treated as array of character type. The type is **size_t** and the macro **NULL**. Various methods are used for determining the lengths of the arrays, but in all cases a **char *** or **void *** argument points to the initial (lowest addressed) character of the array. If an array is accessed beyond the end of an object, the behavior is undefined.

SEE ALSO

American National Standard X3.159-1989

ansiTime

NAME

ansiTime – ANSI time documentation

ROUTINES

asctime() – convert broken-down time into a string (ANSI)
asctime_r() – convert broken-down time into a string (POSIX)
clock() – determine the processor time in use (ANSI)
ctime() – convert time in seconds into a string (ANSI)
ctime_r() – convert time in seconds into a string (POSIX)
difftime() – compute the difference between two calendar times (ANSI)
gmtime() – convert calendar time into UTC broken-down time (ANSI)
gmtime_r() – convert calendar time into broken-down time (POSIX)

localtime() – convert calendar time into broken-down time (ANSI)
localtime_r() – convert calendar time into broken-down time (POSIX)
mktime() – convert broken-down time into calendar time (ANSI)
strftime() – convert broken-down time into a formatted string (ANSI)
time() – determine the current calendar time (ANSI)

DESCRIPTION The header **time.h** defines two macros and declares four types and several functions for manipulating time. Many functions deal with a **calendar time** that represents the current date (according to the Gregorian calendar) and time. Some functions deal with **local time**, which is the calendar time expressed for some specific time zone, and with Daylight Saving Time, which is a temporary change in the algorithm for determining local time. The local time zone and Daylight Saving Time are implementation-defined.

Macros The macros defined are NULL and:

CLOCKS_PER_SEC
the number of ticks per second.

Types The types declared are **size_t** and:

clock_t, time_t
arithmetic types capable of representing times.

struct tm

holds the components of a calendar time in what is known as "broken-down time." The structure contains at least the following members, in any order. The semantics of the members and their normal ranges are expressed in the comments.

```
int tm_sec;    seconds after the minute   - [0, 59]
int tm_min;    minutes after the hour     - [0, 59]
int tm_hour;   hours after midnight      - [0, 23]
int tm_mday;   day of the month          - [1, 31]
int tm_mon;    months since January      - [0, 11]
int tm_year;   years since 1900
int tm_wday;   days since Sunday         - [0, 6]
int tm_yday;   days since January 1     - [0, 365]
int tm_isdst;  Daylight Saving Time flag
```

The value of **tm_isdst** is positive if Daylight Saving Time is in effect, zero if Daylight Saving Time is not in effect, and negative if the information is not available.

If the environment variable TIMEZONE is set, the information is retrieved from this variable, otherwise from the locale information. TIMEZONE is of the form:

name_of_zone:(unused):time_in_minutes_from_UTC:daylight_start:daylight_end

To calculate local time, the value of *time_in_minutes_from_UTC* is subtracted from UTC; *time_in_minutes_from_UTC* must be positive.

arpLib

Daylight information is expressed as mmddhh (month-day-hour), for example:

UTC::0:040102:100102

| | |
|----------------------|---|
| REENTRANCY | Where there is a pair of routines, such as <i>div()</i> and <i>div_r()</i> , only the routine <i>xxx_r()</i> is reentrant. The <i>xxx()</i> routine is not reentrant. |
| INCLUDE FILES | time.h |
| SEE ALSO | ansiLocale , American National Standard X3.159-1989 |

arpLib

| | |
|----------------------|---|
| NAME | arpLib – Address Resolution Protocol (ARP) table manipulation library |
| ROUTINES | <i>arpAdd()</i> – add an entry to the system ARP table <i>arpDelete()</i> – delete an entry from the system ARP table <i>arpFlush()</i> – flush all entries in the system ARP table |
| DESCRIPTION | This library provides functionality for manipulating the system Address Resolution Protocol (ARP) table (cache). ARP is used by the networking modules to map dynamically between Internet Protocol (IP) addresses and physical hardware (Ethernet) addresses. Once these addresses get resolved, they are stored in the system ARP table. Two routines allow the caller to modify this ARP table manually: <i>arpAdd()</i> and <i>arpDelete()</i> . Use <i>arpAdd()</i> to add new or modify existing entries in the ARP table. Use <i>arpDelete()</i> to delete entries from the ARP table. Use <i>arpShow()</i> to show current entries in the ARP table. |
| INCLUDE FILES | arpLib.h |
| SEE ALSO | inetLib , routeLib , etherLib , netShow , <i>VxWorks Programmer's Guide: Network</i> |

ataDrv

| | |
|-----------------|--|
| NAME | ataDrv – ATA/IDE (LOCAL and PCMCIA) disk device driver |
| ROUTINES | <i>ataDrv()</i> – initialize the ATA driver <i>ataDevCreate()</i> – create a device for a ATA/IDE disk <i>ataRawio()</i> – do raw I/O access |

DESCRIPTION This is a driver for ATA/IDE devices on PCMCIA, ISA, and other buses. The driver can be customized via various macros to run on a variety of boards and both big-endian, and little endian CPUs.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. However, two routines must be called directly: *ataDrv()* to initialize the driver and *ataDevCreate()* to create devices.

Before the driver can be used, it must be initialized by calling *ataDrv()*. This routine must be called exactly once, before any reads, writes, or calls to *ataDevCreate()*. Normally, it is called from *usrRoot()* in *usrConfig.c*.

The routine *ataRawio()* supports physical I/O access. The first argument is a drive number, 0 or 1; the second argument is a pointer to an *ATA_RAW* structure.

NOTE Format is not supported, because ATA/IDE disks are already formatted, and bad sectors are mapped.

PARAMETERS The *ataDrv()* function requires a configuration flag as a parameter. The configuration flag is one of the following:

Transfer mode

| | |
|----------------------|----------------------------|
| <i>ATA_PIO_DEF_0</i> | PIO default mode |
| <i>ATA_PIO_DEF_1</i> | PIO default mode, no IORDY |
| <i>ATA_PIO_0</i> | PIO mode 0 |
| <i>ATA_PIO_1</i> | PIO mode 1 |
| <i>ATA_PIO_2</i> | PIO mode 2 |
| <i>ATA_PIO_3</i> | PIO mode 3 |
| <i>ATA_PIO_4</i> | PIO mode 4 |
| <i>ATA_PIO_AUTO</i> | PIO max supported mode |
| <i>ATA_DMA_0</i> | DMA mode 0 |
| <i>ATA_DMA_1</i> | DMA mode 1 |
| <i>ATA_DMA_2</i> | DMA mode 2 |
| <i>ATA_DMA_AUTO</i> | DMA max supported mode |

Transfer bits

| | |
|--------------------|-----------------------|
| <i>ATA_BITS_16</i> | RW bits size, 16 bits |
| <i>ATA_BITS_32</i> | RW bits size, 32 bits |

Transfer unit

| | |
|-----------------------|----------------------|
| <i>ATA_PIO_SINGLE</i> | RW PIO single sector |
| <i>ATA_PIO_MULTI</i> | RW PIO multi sector |
| <i>ATA_DMA_SINGLE</i> | RW DMA single word |
| <i>ATA_DMA_MULTI</i> | RW DMA multi word |

Geometry parameters

| | |
|-------------------------|---------------------------|
| ATA_GEO_FORCE | set geometry in the table |
| ATA_GEO_PHYSICAL | set physical geometry |
| ATA_GEO_CURRENT | set current geometry |

DMA transfer is not supported in this release. If **ATA_PIO_AUTO** or **ATA_DMA_AUTO** is specified, the driver automatically chooses the maximum mode supported by the device. If **ATA_PIO_MULTI** or **ATA_DMA_MULTI** is specified, and the device does not support it, the driver automatically chooses single sector or word mode. If **ATA_BITS_32** is specified, the driver uses 32-bit transfer mode regardless of the capability of the drive.

If **ATA_GEO_PHYSICAL** is specified, the driver uses the physical geometry parameters stored in the drive. If **ATA_GEO_CURRENT** is specified, the driver uses current geometry parameters initialized by BIOS. If **ATA_GEO_FORCE** is specified, the driver uses geometry parameters stored in **sysLib.c**.

The geometry parameters are stored in the structure table **ataTypes[]** in **sysLib.c**. That table has two entries, the first for drive 0, the second for drive 1. The members of the structure are:

```

int cylinders;           /* number of cylinders */
int heads;              /* number of heads */
int sectors;           /* number of sectors per track */
int bytes;             /* number of bytes per sector */
int precomp;          /* precompensation cylinder */

```

This driver does not access the PCI-chip-set IDE interface, but rather takes advantage of BIOS or VxWorks initialization. Thus, the BIOS setting should match the modes specified by the configuration flag.

The BSP may provide a **sysAtaInit()** routine for situations where an ATA controller RESET (0x1f6 or 0x3f6, bit 2 is set) clears ATA specific functionality in a chipset that is not re-enabled per the ATA-2 spec.

This BSP routine should be declared in **sysLib.c** or **sysAta.c** as follows:

```

void sysAtaInit (BOOL ctrl)
{
    /* BSP SPECIFIC CODE HERE */
}

```

Then the BSP should perform the following operation before **ataDrv()** is called, in **sysHwInit** for example:

```

IMPORT VOIDFUNCPTR _func_sysAtaInit;
/* setup during initialization */
_func_sysAtaInit = (VOIDFUNCPTR) sysAtaInit;

```

It should contain chipset specific reset code, such as code which re-enables PCI write posting for an integrated PCI-IDE device, for example. This will be executed during every

ataDrv(), *ataInit()*, and *ataReset()* or equivalent block device routine. If the *sysAtaInit* routine is not provided by the BSP it is ignored by the driver, therefore it is not a required BSP routine.

SEE ALSO *VxWorks Programmer's Guide: I/O System*

ataShow

NAME *ataShow* – ATA/IDE (LOCAL and PCMCIA) disk device driver show routine

ROUTINES *ataShowInit()* – initialize the ATA/IDE disk driver show routine
ataShow() – show the ATA/IDE disk parameters

DESCRIPTION This library contains a driver show routine for the ATA/IDE (PCMCIA and LOCAL) devices supported on the IBM PC.

bALib

NAME *bALib* – buffer manipulation library SPARC assembly language routines

ROUTINES *bzeroDoubles()* – zero out a buffer eight bytes at a time (SPARC)
bfillDoubles() – fill a buffer with a specified eight-byte pattern (SPARC)
bcopyDoubles() – copy one buffer to another eight bytes at a time (SPARC)

DESCRIPTION This library contains routines to manipulate buffers, which are simply variable length byte arrays. These routines are highly optimized loops.

All address pointers must be properly aligned for 8-byte moves. Note that buffer lengths are specified in terms of bytes or doubles. Since this is meant to be a high-performance operation, the minimum number of bytes is 256.

NOTE None of the buffer routines have been hand-coded in assembly. These are additional routines that exploit the SPARC's LDD and STD instructions.

SEE ALSO *bLib*, *ansiString*

bLib

| | |
|----------------------|--|
| NAME | bLib – buffer manipulation library |
| ROUTINES | <i>bcmp()</i> – compare one buffer to another <i>binvert()</i> – invert the order of bytes in a buffer <i>bswap()</i> – swap buffers <i>swab()</i> – swap bytes <i>uswab()</i> – swap bytes with buffers that are not necessarily aligned <i>bzero()</i> – zero out a buffer <i>bcopy()</i> – copy one buffer to another <i>bcopyBytes()</i> – copy one buffer to another one byte at a time <i>bcopyWords()</i> – copy one buffer to another one word at a time <i>bcopyLongs()</i> – copy one buffer to another one long word at a time <i>bfill()</i> – fill a buffer with a specified character <i>bfillBytes()</i> – fill buffer with a specified character one byte at a time <i>index()</i> – find the first occurrence of a character in a string <i>rindex()</i> – find the last occurrence of a character in a string |
| DESCRIPTION | <p>This library contains routines to manipulate buffers of variable-length byte arrays. Operations are performed on long words when possible, even though the buffer lengths are specified in bytes. This occurs only when source and destination buffers start on addresses that are both odd or both even. If one buffer is even and the other is odd, operations must be done one byte at a time (because of alignment problems inherent in the MC68000), thereby slowing down the process.</p> <p>Certain applications, such as byte-wide memory-mapped peripherals, may require that only byte operations be performed. For this purpose, the routines <i>bcopyBytes()</i> and <i>bfillBytes()</i> provide the same functions as <i>bcopy()</i> and <i>bfill()</i>, but use only byte-at-a-time operations. These routines do not check for null termination.</p> |
| INCLUDE FILES | string.h |
| SEE ALSO | ansiString |

bootConfig

| | |
|--------------------|--|
| NAME | bootConfig – system configuration module for boot ROMs |
| ROUTINES | No Callable Routines |
| DESCRIPTION | This is the WRS-supplied configuration module for the VxWorks boot ROM. It is a stripped-down version of usrConfig.c , having no VxWorks shell or debugging facilities. Its primary function is to load an object module over the network with either RSH or FTP. Additionally, a simple set of single letter commands is provided for displaying and modifying memory contents. Use this module as a starting point for placing applications in ROM. |

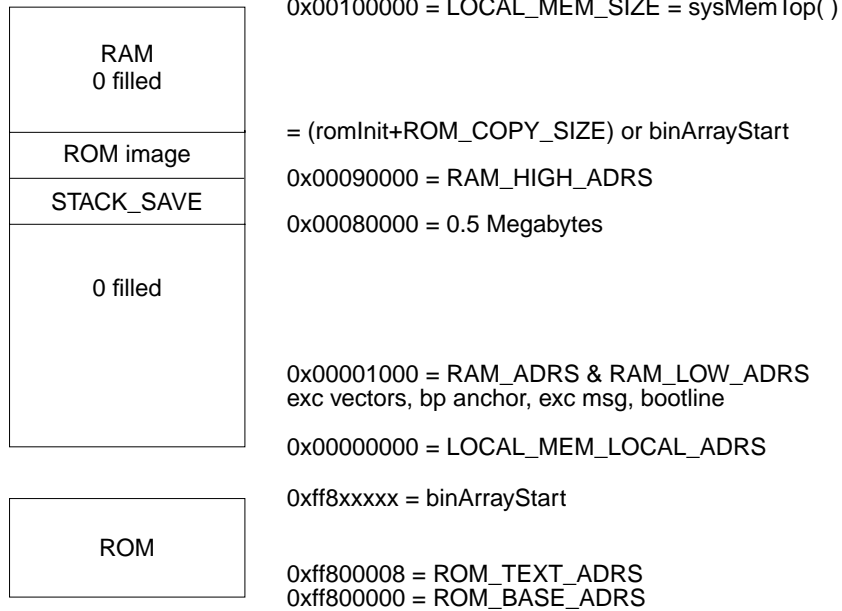
bootInit

| | |
|--------------------|--|
| NAME | bootInit – ROM initialization module |
| ROUTINES | <i>romStart()</i> – generic ROM initialization |
| DESCRIPTION | <p>This module provides a generic boot ROM facility. The target-specific romInit.s module performs the minimal preliminary board initialization and then jumps to the C routine <i>romStart()</i>. This routine, still executing out of ROM, copies the first stage of the startup code to a RAM address and jumps to it. The next stage clears memory and then uncompresses the remainder of ROM into the final VxWorks ROM image in RAM.</p> <p>A modified version of the Public Domain zlib library is used to uncompress the VxWorks boot ROM executable linked with it. Compressing object code typically achieves over 55% compression, permitting much larger systems to be burned into ROM. The only expense is the added few seconds delay while the first two stages complete.</p> |

ROM AND RAM MEMORY LAYOUT

Example memory layout for a 1-megabyte board:

bootLib



SEE ALSO *inflate()*, *romInit()*, *deflate*

AUTHOR The original compression software for zlib was written by Jean-loup Gailly and Mark Adler. See the reference pages for *inflate()* and *deflate* for more information on their freely available compression software.

bootLib

NAME *bootLib* – boot ROM subroutine library

ROUTINES

- bootStringToStruct()* – interpret the boot parameters from the boot line
- bootStructToString()* – construct a boot line
- bootParamsShow()* – display boot line parameters
- bootParamsPrompt()* – prompt for boot line parameters
- bootLeaseExtract()* – extract the lease information from an Internet address
- bootNetmaskExtract()* – extract the net mask field from an Internet address
- bootBpAnchorExtract()* – extract a backplane address from a device field

DESCRIPTION

This library contains routines for manipulating a boot line. Routines are provided to interpret, construct, print, and prompt for a boot line.

When VxWorks is first booted, certain parameters can be specified, such as network addresses, boot device, host, and start-up file. This information is encoded into a single ASCII string known as the boot line. The boot line is placed at a known address (specified in **config.h**) by the boot ROMs so that the system being booted can discover the parameters that were used to boot the system. The boot line is the only means of communication from the boot ROMs to the booted system.

The boot line is of the form:

```
bootdev(unitnum,procnum)hostname:filename e=# b=# h=# g=# u=userid pw=passwd  
f=# tn=targetname s=startupscript o=other
```

bootdev

the boot device (required); for example, "ex" for Excelan Ethernet, "bp" for backplane. For the backplane, this field can have an optional anchor address specification of the form "bp=*adrs*" (see *bootBpAnchorExtract()*).

unitnum

the unit number of the boot device (0..n).

procnum

the processor number on the backplane, 0..n (required for VME boards).

hostname

the name of the boot host (required).

filename

the file to be booted (required).

- e** the Internet address of the Ethernet interface. This field can have an optional subnet mask of the form *inet_adrs:subnet_mask*. If DHCP is used to obtain the configuration parameters, lease timing information may also be present. This information takes the form *lease_duration:lease_origin* and is appended to the end of the field. (see *bootNetmaskExtract()* and *bootLeaseExtract()*).
- b** the Internet address of the backplane interface. This field can have an optional subnet mask and/or lease timing information as "e".
- h** the Internet address of the boot host.
- g** the Internet address of the gateway to the boot host. Leave this parameter blank if the host is on same network.
- u** a valid user name on the boot host.
- pw** the password for the user on the host. This parameter is usually left blank. If specified, FTP is used for file transfers.
- f** the system-dependent configuration flags. This parameter contains an **or** of option bits defined in **sysLib.h**.

bootpLib

- tn** the name of the system being booted
- s** the name of a file to be executed as a start-up script.
- o** "other" string for use by the application.

The Internet addresses are specified in "dot" notation (e.g., 90.0.0.2). The order of assigned values is arbitrary.

EXAMPLE

```
enp(0,0)host:/usr/wpwr/target/config/mz7122/vxWorks e=90.0.0.2 b=91.0.0.2
h=100.0.0.4 g=90.0.0.3 u=bob pw=realtime f=2 tn=target
s=host:/usr/bob/startup o=any_string
```

INCLUDE FILES **bootLib.h**

SEE ALSO **bootConfig**

bootpLib

NAME **bootpLib** – BOOTP client library

ROUTINES *bootpParamsGet()* – retrieve boot parameters using BOOTP
bootpMsgSend() – send a BOOTP request message

DESCRIPTION This library implements the client side of the Bootstrap Protocol (BOOTP). This network protocol allows the dynamic configuration of the target's boot parameters at boot time. This is in contrast to using the boot information encoded in system non-volatile RAM or ROM. Thus, at boot time, BOOTP goes over the network to get an IP address, a boot file name, and the boot host's IP address.

The actual transfer of the boot image is handled by a file transfer protocol, such as TFTP or FTP, or by an RSH command.

To access BOOTP services, you can use either the high-level interface supported by *bootpParamsGet()*, or the low-level interface supported by *bootpMsgSend()*.

HIGH-LEVEL INTERFACE

The *bootpParamsGet()* routine provides the highest level interface to BOOTP. It accepts a parameter descriptor structure that allows the retrieval of any combination of the options described in RFC 1533 (if supported by the BOOTP server and if specified in the database). During system boot, the routine obtains the boot file, the Internet address, and the host Internet address. It also obtains the subnet mask and the Internet address of an IP router, if available.

LOW-LEVEL INTERFACE

The *bootpMsgSend()* routine provides a lower-level interface to BOOTP. It accepts and returns a BOOTP message as a parameter. This interface is more flexible because it gives the caller direct access to the data in the BOOTP request/reply messages. For example, if the BOOTP message includes implementation-specific options not defined in an RFC, the caller can use *bootpMsgSend()* to retrieve them from the vendor-specific field in the BOOTP message. The *bootpParamsGet()* routine already provides all defined options.

EXAMPLE

The following code provides an example of how to use *bootpParamsGet()*:

```
#include "bootpLib.h"
struct bootpParams bootParams;
struct in_addr      clntAddr;
struct in_addr      hostAddr;
char                bootFile [FILENAME_SIZE];
int                 subnetMask;
struct in_addr_list routerList;
struct in_addr      gateway;
char                clntAddr [INET_ADDR_LEN];
char                bootServer [INET_ADDR_LEN];
char                bootFile [SIZE_FILE];
int                 fileSize;
int                 subnetMask;
char                gateway [INET_ADDR_LEN];

bzero ( (char *)&clntAddr, sizeof (struct in_addr));
bzero ( (char *)&hostAddr, sizeof (struct in_addr));
bzero (bootFile, FILENAME_SIZE);
subnetMask = 0;
bzero ( (char *)&gateway, sizeof (struct in_addr));
/* Set all pointers in parameter descriptor to NULL. */
bzero ((char *)&bootParams, sizeof (struct bootpParams));
/* Set pointers corresponding to desired options. */
bootParams.clientAddr = &clntAddr;
bootParams.bootHostAddr = &hostAddr;
bootParams.bootfile = pBootFile;
bootParams.netmask = (struct in_addr *)&subnetMask;
routerlist.addr = &gateway;
routerlist.num = 1;
bootParams.routers = &routerlist;
if (bootpParamsGet ("ln0", 0, 0, &bootParams) == ERROR)
    return (ERROR);
```

NOTE

Certain targets (typically those with no NVRAM) construct their Ethernet address based on the target's IP address. An IP address must be entered for these targets in order to boot over the network. The remaining information can be obtained with BOOTP.

cacheArchLib

BOOTP is not supported over the following network interfaces: if_sl (SLIP) and if_ie (Sun IE driver). if_sl (SLIP) and if_ppp (PPP).

INCLUDE FILES **bootpLib.h**

SEE ALSO **bootLib**, RFC 951, RFC 1542, RFC 1533, *VxWorks Programmer's Guide: Network*

cacheArchLib

NAME **cacheArchLib** – architecture-specific cache management library

ROUTINES **cacheArchLibInit()** – initialize the cache library
cacheArchClearEntry() – clear an entry from a cache (68K, x86)
cacheStoreBufEnable() – enable the store buffer (MC68060 only)
cacheStoreBufDisable() – disable the store buffer (MC68060 only)

DESCRIPTION This library contains architecture-specific cache library functions for the following processor cache families: Motorola 68K, Intel 960, Intel x86, PowerPC, ARM, and the Solaris, HP-UX, and NT simulators. Each routine description indicates which architecture families support it. Within families, different members support different cache mechanisms; thus, some operations cannot be performed by certain processors because they lack particular functionalities. In such cases, the routines in this library return ERROR. Processor-specific constraints are addressed in the manual entries for routines in this library. If the caches are unavailable or uncontrollable, the routines return ERROR. The exception to this rule is the 68020; although the 68020 has no cache, data cache operations return OK.

The SPARC and MIPS architecture families have cache-related routines in individual BSP libraries. See the reference pages for the individual libraries and routines.

INCLUDE FILES **cacheLib.h**, **mmuLib.h** (ARM only)

SEE ALSO **cacheLib**, **vmLib**

cacheCy604Lib

| | |
|----------------------|--|
| NAME | cacheCy604Lib – Cypress CY7C604/605 SPARC cache management library |
| ROUTINES | <i>cacheCy604LibInit()</i> – initialize the Cypress CY7C604 cache library <i>cacheCy604ClearLine()</i> – clear a line from a CY7C604 cache <i>cacheCy604ClearPage()</i> – clear a page from a CY7C604 cache <i>cacheCy604ClearSegment()</i> – clear a segment from a CY7C604 cache <i>cacheCy604ClearRegion()</i> – clear a region from a CY7C604 cache |
| DESCRIPTION | <p>This library contains architecture-specific cache library functions for the Cypress CY7C604 architecture. There is a 64-Kbyte mixed instruction and data cache that operates in write-through or copyback mode. Each cache line contains 32 bytes. Cache tag operations are performed with "line," "page," "segment," or "region" granularity.</p> <p>MMU (Memory Management Unit) support is needed to mark pages cacheable or non-cacheable. For more information, see the manual entry for vmLib.</p> <p>For general information about caching, see the manual entry for cacheLib.</p> |
| INCLUDE FILES | cacheLib.h |
| SEE ALSO | cacheLib , vmLib |

cacheI960CxALib

| | |
|----------------------|--|
| NAME | cacheI960CxALib – I960Cx cache management assembly routines |
| ROUTINES | <i>cacheI960CxICDisable()</i> – disable the I960Cx instruction cache (i960) <i>cacheI960CxICEnable()</i> – enable the I960Cx instruction cache (i960) <i>cacheI960CxICInvalidate()</i> – invalidate the I960Cx instruction cache (i960) <i>cacheI960CxICLoadNLock()</i> – load and lock I960Cx 512-byte instruction cache (i960) <i>cacheI960CxIC1kLoadNLock()</i> – load and lock I960Cx 1KB instruction cache (i960) |
| DESCRIPTION | <p>This library contains Intel I960Cx cache management routines written in assembly language. The I960CX utilize a 1KB instruction cache and no data cache.</p> <p>For general information about caching, see the manual entry for cacheLib.</p> |
| INCLUDE FILES | cacheLib.h |
| SEE ALSO | cacheI960CxLib , cacheLib , <i>I960Cx Processors User's Manual</i> |

cacheI960CxLib

| | |
|----------------------|--|
| NAME | cacheI960CxLib – I960Cx cache management library |
| ROUTINES | <i>cacheI960CxLibInit()</i> – initialize the I960Cx cache library (i960) |
| DESCRIPTION | <p>This library contains architecture-specific cache library functions for the Intel I960Cx architecture. The I960Cx utilizes a 1KB instruction cache and no data cache. Cache line size is fixed at 16 bytes.</p> <p>For general information about caching, see the manual entry for cacheLib.</p> |
| INCLUDE FILES | cacheLib.h |
| SEE ALSO | cacheLib , <i>Intel i960Cx User's Manual</i> |

cacheI960JxA Lib

| | |
|--------------------|--|
| NAME | cacheI960JxA Lib – I960Jx cache management assembly routines |
| ROUTINES | <p><i>cacheI960JxICDisable()</i> – disable the I960Jx instruction cache (i960) <i>cacheI960JxICEnable()</i> – enable the I960Jx instruction cache (i960) <i>cacheI960JxICInvalidate()</i> – invalidate the I960Jx instruction cache (i960) <i>cacheI960JxICLoadNLock()</i> – load and lock the I960Jx instruction cache (i960) <i>cacheI960JxICStatusGet()</i> – get the I960Jx instruction cache status (i960) <i>cacheI960JxICLockingStatusGet()</i> – get the I960Jx I-cache locking status (i960) <i>cacheI960JxICFlush()</i> – flush the I960Jx instruction cache (i960) <i>cacheI960JxDCDisable()</i> – disable the I960Jx data cache (i960) <i>cacheI960JxDCEnable()</i> – enable the I960Jx data cache (i960) <i>cacheI960JxDCInvalidate()</i> – invalidate the I960Jx data cache (i960) <i>cacheI960JxDCoherent()</i> – ensure data cache coherency (i960) <i>cacheI960JxDCStatusGet()</i> – get the I960Jx data cache status (i960) <i>cacheI960JxDCFlush()</i> – flush the I960Jx data cache (i960)</p> |
| DESCRIPTION | <p>This library contains Intel I960Jx cache-management routines written in assembly language. The I960JF and JD utilize a 4KB instruction cache and a 2KB data cache while the I960JA has a 2KB instruction cache and a 1KB data cache that operate in write-through mode.</p> |

Cache line size is fixed at 16 bytes. Cache tags may be invalidated on a per-line basis by execution of a store to a specified line while the cache is in invalidate mode. See also the manual entry for **cacheI960JxLib**.

For general information about caching, see the manual entry for **cacheLib**.

INCLUDE FILES **arch/i960/cacheI960JxLib.h, cacheLib.h**

SEE ALSO **cacheI960JxLib, cacheLib**, *I960Jx Processors User's Manual*

cacheI960JxLib

NAME **cacheI960JxLib** – I960Jx cache management library

ROUTINES *cacheI960JxLibInit()* – initialize the I960Jx cache library (i960)

DESCRIPTION This library contains architecture-specific cache library functions for the Intel I960Jx architecture. The I960JF utilizes a 4KB instruction cache and a 2KB data cache that operate in write-through mode. The I960JA utilizes a 2KB instruction cache and a 1KB data cache that operate in write-through mode. Cache line size is fixed at 16 bytes.

For general information about caching, see the manual entry for **cacheLib**.

INCLUDE FILES **arch/i960/cacheI960JxLib.h, cacheLib.h**

SEE ALSO **cacheLib**, *Intel i960Jx User's Manual*

cacheLib

NAME **cacheLib** – cache management library

ROUTINES *cacheLibInit()* – initialize the cache library for a processor architecture
cacheEnable() – enable the specified cache
cacheDisable() – disable the specified cache
cacheLock() – lock all or part of a specified cache
cacheUnlock() – unlock all or part of a specified cache
cacheFlush() – flush all or some of a specified cache
cacheInvalidate() – invalidate all or some of a specified cache
cacheClear() – clear all or some entries from a cache
cachePipeFlush() – flush processor write buffers to memory

cacheLib

cacheTextUpdate() – synchronize the instruction and data caches
cacheDmaMalloc() – allocate a cache-safe buffer for DMA devices and drivers
cacheDmaFree() – free the buffer acquired with *cacheDmaMalloc()*
cacheDroFlush() – flush the data cache for drivers
cacheDroInvalidate() – invalidate data cache for drivers
cacheDroVirtToPhys() – translate a virtual address for drivers
cacheDroPhysToVirt() – translate a physical address for drivers

DESCRIPTION

This library provides architecture-independent routines for managing the instruction and data caches. Architecture-dependent routines are documented in the architecture-specific libraries.

The cache library is initialized by *cacheLibInit()* in *usrInit()*. The *cacheLibInit()* routine typically calls an architecture-specific initialization routine in one of the architecture-specific libraries. The initialization routine places the cache in a known and quiescent state, ready for use, but not yet enabled. Cache devices are enabled and disabled by calls to *cacheEnable()* and *cacheDisable()*, respectively.

The structure **CACHE_LIB** in **cacheLib.h** provides a function pointer that allows for the installation of different cache implementations in an architecture-independent manner. If the processor family allows more than one cache implementation, the board support package (BSP) must select the appropriate cache library using the function pointer **sysCacheLibInit**. The *cacheLibInit()* routine calls the initialization function attached to **sysCacheLibInit** to perform the actual **CACHE_LIB** function pointer initialization (see **cacheLib.h**). Note that **sysCacheLibInit** must be initialized when declared; it need not exist for architectures with a single cache design. Systems without caches have all NULL pointers in the **CACHE_LIB** structure. For systems with bus snooping, NULLifying the flush and invalidate function pointers in *sysHwInit()* improves overall system and driver performance.

Function pointers also provide a way to supplement the cache library or attach user-defined cache functions for managing secondary cache systems.

Parameters specified by *cacheLibInit()* are used to select the cache mode, either write-through (**CACHE_WRITETHROUGH**) or copyback (**CACHE_COPYBACK**), as well as to implement all other cache configuration features via software bit-flags. Note that combinations, such as setting copyback and write-through at the same time, do not make sense.

Typically, the first argument passed to cache routines after initialization is the **CACHE_TYPE**, which selects the data cache (**DATA_CACHE**) or the instruction cache (**INSTRUCTION_CACHE**).

Several routines accept two additional arguments: an address and the number of bytes. Some cache operations can be applied to the entire cache (bytes = **ENTIRE_CACHE**) or to a portion of the cache. This range specification allows the cache to be selectively locked, unlocked, flushed, invalidated, and cleared. The two complementary routines, *cacheDmaMalloc()* and *cacheDmaFree()*, are tailored for efficient driver writing. The

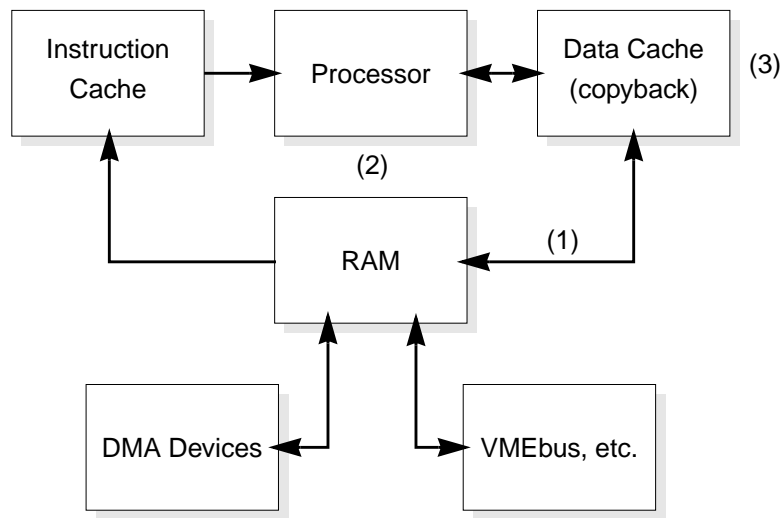
cacheDmaMalloc() routine attempts to return a "cache-safe" buffer, which is created by the MMU and a set of flush and invalidate function pointers. Examples are provided below in the section "Using the Cache Library."

Most routines in this library return a STATUS value of OK, or ERROR if the cache selection is invalid or the cache operation fails.

BACKGROUND

The emergence of RISC processors and effective CISC caches has made cache and MMU support a key enhancement to VxWorks. (For more information about MMU support, see the manual entry for vmLib.) The VxWorks cache strategy is to maintain coherency between the data cache and RAM and between the instruction and data caches. VxWorks also preserves overall system performance. The product is designed to support several architectures and board designs, to have a high-performance implementation for drivers, and to make routines functional for users, as well as within the entire operating system. The lack of a consistent cache design, even within architectures, has required designing for the case with the greatest number of coherency issues (Harvard architecture, copyback mode, DMA devices, multiple bus masters, and no hardware coherency support).

Caches run in two basic modes, write-through and copyback. The write-through mode forces all writes to the cache and to RAM, providing partial coherency. Writing to RAM every time, however, slows down the processor and uses bus bandwidth. The copyback mode conserves processor performance time and bus bandwidth by writing only to the cache, not RAM. Copyback cache entries are only written to memory on demand. A Least Recently Used (LRU) algorithm is typically used to determine which cache line to displace and flush. Copyback provides higher system performance, but requires more coherency support. Below is a logical diagram of a cached system to aid in the visualization of the coherency issues.



The loss of cache coherency for a VxWorks system occurs in three places:

- (1) data cache / RAM
- (2) instruction cache / data cache
- (3) shared cache lines

A problem between the data cache and RAM (1) results from asynchronous accesses (reads and writes) to the RAM by the processor and other masters. Accesses by DMA devices and alternate bus masters (shared memory) are the primary causes of incoherency, which can be remedied with minor code additions to the drivers.

The instruction cache and data cache (2) can get out of sync when the loader, the debugger, and the interrupt connection routines are being used. The instructions resulting from these operations are loaded into the data cache, but not necessarily the instruction cache, in which case there is a coherency problem. This can be fixed by "flushing" the data cache entries to RAM, then "invalidating" the instruction cache entries. The invalid instruction cache tags will force the retrieval of the new instructions that the data cache has just flushed to RAM.

Cache lines that are shared (3) by more than one task create coherency problems. These are manifest when one thread of execution invalidates a cache line in which entries may belong to another thread. This can be avoided by allocating memory on a cache line boundary, then rounding up to a multiple of the cache line size.

The best way to preserve cache coherency with optimal performance (Harvard architecture, copyback mode, no software intervention) is to use hardware with bus snooping capabilities. The caches, the RAM, the DMA devices, and all other bus masters are tied to a physical bus where the caches can "snoop" or watch the bus transactions. The address cycle and control (read/write) bits are broadcast on the bus to allow snooping. Data transfer cycles are deferred until absolutely necessary. When one of the entries on the physical side of the cache is modified by an asynchronous action, the cache(s) marks its entry(s) as invalid. If an access is made by the processor (logical side) to the now invalid cached entry, it is forced to retrieve the valid entry from RAM. If while in copyback mode the processor writes to a cached entry, the RAM version becomes stale. If another master attempts to access that stale entry in RAM, the cache with the valid version pre-empts the access and writes the valid data to RAM. The interrupted access then restarts and retrieves the now-valid data in RAM. Note that this configuration allows only one valid entry at any time. At this time, only a few boards provide the snooping capability; therefore, cache support software must be designed to handle incoherency hazards without degrading performance.

The determinism, interrupt latency, and benchmarks for a cached system are exceedingly difficult to specify (best case, worst case, average case) due to cache hits and misses, line flushes and fills, atomic burst cycles, global and local instruction and data cache locking, copyback versus write-through modes, hardware coherency support (or lack of), and MMU operations (table walks, TLB locking).

USING THE CACHE LIBRARY

The coherency problems described above can be overcome by adding cache support to existing software. For code segments that are not time-critical (loader, debugger, interrupt connection), the following sequence should be used first to flush the data cache entries and then to invalidate the corresponding instruction cache entries.

```
cacheFlush (DATA_CACHE, address, bytes);  
cacheInvalidate (INSTRUCTION_CACHE, address, bytes);
```

For time-critical code, implementation is up to the driver writer. The following are tips for using the VxWorks cache library effectively.

Incorporate cache calls in the driver program to maintain overall system performance. The cache may be disabled to facilitate driver development; however, high-performance production systems should operate with the cache enabled. A disabled cache will dramatically reduce system performance for a completed application.

Buffers can be static or dynamic. Mark buffers "non-cacheable" to avoid cache coherency problems. This usually requires MMU support. Dynamic buffers are typically smaller than their static counterparts, and they are allocated and freed often. When allocating either type of buffer, it should be designated non-cacheable; however, dynamic buffers should be marked "cacheable" before being freed. Otherwise, memory becomes fragmented with numerous non-cacheable dynamic buffers.

Alternatively, use the following flush/invalidate scheme to maintain cache coherency.

```
cacheInvalidate (DATA_CACHE, address, bytes); /* input buffer */  
cacheFlush (DATA_CACHE, address, bytes); /* output buffer */
```

The principle is to flush output buffers before each use and invalidate input buffers before each use. Flushing only writes modified entries back to RAM, and instruction cache entries never get modified.

Several flush and invalidate macros are defined in **cacheLib.h**. Since optimized code uses these macros, they provide a mechanism to avoid unnecessary cache calls and accomplish the necessary work (return OK). Needless work includes flushing a write-through cache, flushing or invalidating cache entries in a system with bus snooping, and flushing or invalidating cache entries in a system without caches. The macros are set to reflect the state of the cache system hardware and software. Example 1 The following example is of a simple driver that uses *cacheFlush()* and *cacheInvalidate()* from the cache library to maintain coherency and performance. There are two buffers (lines 3 and 4), one for input and one for output. The output buffer is obtained by the call to *memalign()*, a special version of the well-known *malloc()* routine (line 6). It returns a pointer that is rounded down and up to the alignment parameter's specification. Note that cache lines should not be shared, therefore *_CACHE_ALIGN_SIZE* is used to force alignment. If the memory allocator fails (line 8), the driver will typically return ERROR (line 9) and quit.

The driver fills the output buffer with initialization information, device commands, and data (line 11), and is prepared to pass the buffer to the device. Before doing so the driver must flush the data cache (line 13) to ensure that the buffer is in memory, not hidden in

cacheLib

the cache. The *drvWrite()* routine lets the device know that the data is ready and where in memory it is located (line 14).

More driver code is executed (line 16), then the driver is ready to receive data that the device has placed in an input buffer in memory (line 18). Before the driver can work with the incoming data, it must invalidate the data cache entries (line 19) that correspond to the input buffer's data in order to eliminate stale entries. That done, it is safe for the driver to retrieve the input data from memory (line 21). Remember to free (line 23) the buffer acquired from the memory allocator. The driver will return OK (line 24) to distinguish a successful from an unsuccessful operation.

```

STATUS drvExample1 ()           /* simple driver, good performance */
{
3: void *      pInBuf;          /* input buffer */
4: void *      pOutBuf;        /* output buffer */

6: pOutBuf = memalign (_CACHE_ALIGN_SIZE, BUF_SIZE);
8: if (pOutBuf == NULL)
9:     return (ERROR);        /* memory allocator failed */
11: /* other driver initialization and buffer filling */
13: cacheFlush (DATA_CACHE, pOutBuf, BUF_SIZE);
14: drvWrite (pOutBuf);        /* output data to device */
16: /* more driver code */
18: pInBuf = drvRead ();       /* wait for device data */
19: cacheInvalidate (DATA_CACHE, pInBuf, BUF_SIZE);
21: /* handle input data from device */
23: free (pOutBuf);           /* return buffer to memory pool */
24: return (OK);
}

```

Extending this flush/invalidate concept further, individual buffers can be treated this way, not just the entire cache system. The idea is to avoid unnecessary flush and/or invalidate operations on a per-buffer basis by allocating cache-safe buffers. Calls to *cacheDmaMalloc()* optimize the flush and invalidate function pointers to NULL, if possible, while maintaining data integrity. Example 2 The following example is of a high-performance driver that takes advantage of the cache library to maintain coherency. It uses *cacheDmaMalloc()* and the macros *CACHE_DMA_FLUSH* and *CACHE_DMA_INVALIDATE*. A buffer pointer is passed as a parameter (line 2). If the pointer is not NULL (line 7), it is assumed that the buffer will not experience any cache coherency problems. If the driver was not provided with a cache-safe buffer, it will get one (line 11) from *cacheDmaMalloc()*. A *CACHE_FUNCS* structure (see *cacheLib.h*) is used to create a buffer that will not suffer from cache coherency problems. If the memory allocator fails (line 13), the driver will typically return ERROR (line 14) and quit.

The driver fills the output buffer with initialization information, device commands, and data (line 17), and is prepared to pass the buffer to the device. Before doing so, the driver must flush the data cache (line 19) to ensure that the buffer is in memory, not hidden in

the cache. The routine *drvWrite()* lets the device know that the data is ready and where in memory it is located (line 20).

More driver code is executed (line 22), and the driver is then ready to receive data that the device has placed in the buffer in memory (line 24). Before the driver cache can work with the incoming data, it must invalidate the data cache entries (line 25) that correspond to the input buffer's data in order to eliminate stale entries. That done, it is safe for the driver to handle the input data (line 27), which the driver retrieves from memory. Remember to free the buffer (line 29) acquired from the memory allocator. The driver will return OK (line 30) to distinguish a successful from an unsuccessful operation.

```

STATUS drvExample2 (pBuf)          /* simple driver, great performance */
2: void *      pBuf;              /* buffer pointer parameter */
    {
5:  if (pBuf != NULL)
        {
7:      /* no cache coherency problems with buffer passed to driver */
        }
    else
        {
11:     pBuf = cacheDmaMalloc (BUF_SIZE);
13:     if (pBuf == NULL)
14:         return (ERROR);      /* memory allocator failed */
        }
17: /* other driver initialization and buffer filling */
19: CACHE_DMA_FLUSH (pBuf, BUF_SIZE);
20: drvWrite (pBuf);             /* output data to device */
22: /* more driver code */
24: drvWait ();                  /* wait for device data */
25: CACHE_DMA_INVALIDATE (pBuf, BUF_SIZE);
27: /* handle input data from device */
29: cacheDmaFree (pBuf);         /* return buffer to memory pool */
30: return (OK);
    }

```

Do not use `CACHE_DMA_FLUSH` or `CACHE_DMA_INVALIDATE` without first calling *cacheDmaMalloc()*, otherwise the function pointers may not be initialized correctly. Note that this driver scheme assumes all cache coherency modes have been set before driver initialization, and that the modes do not change after driver initialization. The *cacheFlush()* and *cacheInvalidate()* functions can be used at any time throughout the system since they are affiliated with the hardware, not the malloc/free buffer.

A call to *cacheLibInit()* in write-through mode makes the flush function pointers NULL. Setting the caches in copyback mode (if supported) should set the pointer to and call an architecture-specific flush routine. The invalidate and flush macros may be NULLified if the hardware provides bus snooping and there are no cache coherency problems. Example 3 The next example shows a more complex driver that requires address

cacheLib

translations to assist in the cache coherency scheme. The previous example had **a priori** knowledge of the system memory map and/or the device interaction with the memory system. This next driver demonstrates a case in which the virtual address returned by `cacheDmaMalloc()` might differ from the physical address seen by the device. It uses the `CACHE_DMA_VIRT_TO_PHYS` and `CACHE_DMA_PHYS_TO_VIRT` macros in addition to the `CACHE_DMA_FLUSH` and `CACHE_DMA_INVALIDATE` macros.

The `cacheDmaMalloc()` routine initializes the buffer pointer (line 3). If the memory allocator fails (line 5), the driver will typically return `ERROR` (line 6) and quit. The driver fills the output buffer with initialization information, device commands, and data (line 8), and is prepared to pass the buffer to the device. Before doing so, the driver must flush the data cache (line 10) to ensure that the buffer is in memory, not hidden in the cache. The flush is based on the virtual address since the processor filled in the buffer. The `drvWrite()` routine lets the device know that the data is ready and where in memory it is located (line 11). Note that the `CACHE_DMA_VIRT_TO_PHYS` macro converts the buffer's virtual address to the corresponding physical address for the device.

More driver code is executed (line 13), and the driver is then ready to receive data that the device has placed in the buffer in memory (line 15). Note the use of the `CACHE_DMA_PHYS_TO_VIRT` macro on the buffer pointer received from the device. Before the driver cache can work with the incoming data, it must invalidate the data cache entries (line 16) that correspond to the input buffer's data in order to eliminate stale entries. That done, it is safe for the driver to handle the input data (line 17), which it retrieves from memory. Remember to free (line 19) the buffer acquired from the memory allocator. The driver will return `OK` (line 20) to distinguish a successful from an unsuccessful operation.

```

STATUS drvExample3 ( )           /* complex driver, great performance */ {
3: void * pBuf = cacheDmaMalloc (BUF_SIZE);
5: if (pBuf == NULL)
6:     return (ERROR);          /* memory allocator failed */
8: /* other driver initialization and buffer filling */
10: CACHE_DMA_FLUSH (pBuf, BUF_SIZE);
11: drvWrite (CACHE_DMA_VIRT_TO_PHYS (pBuf));
13: /* more driver code */
15: pBuf = CACHE_DMA_PHYS_TO_VIRT (drvRead ());
16: CACHE_DMA_INVALIDATE (pBuf, BUF_SIZE);
17: /* handle input data from device */
19: cacheDmaFree (pBuf);        /* return buffer to memory pool */
20: return (OK);
    }

```

Driver Summary The virtual-to-physical and physical-to-virtual function pointers associated with `cacheDmaMalloc()` are supplements to a cache-safe buffer. Since the processor operates on virtual addresses and the devices access physical addresses, discrepant addresses can occur and might prevent DMA-type devices from being able to access the allocated buffer. Typically, the MMU is used to return a buffer that has pages

marked as non-cacheable. An MMU is used to translate virtual addresses into physical addresses, but it is not guaranteed that this will be a "transparent" translation.

When `cacheDmaMalloc()` does something that makes the virtual address different from the physical address needed by the device, it provides the translation procedures. This is often the case when using translation lookaside buffers (TLB) or a segmented address space to inhibit caching (e.g., by creating a different virtual address for the same physical space.) If the virtual address returned by `cacheDmaMalloc()` is the same as the physical address, the function pointers are made NULL so that no calls are made when the macros are expanded. Board Support Packages Each board for an architecture with more than one cache implementation has the potential for a different cache system. Hence the BSP for selecting the appropriate cache library. The function pointer `sysCacheLibInit` is set to `cacheXxxLibInit()` ("Xxx" refers to the chip-specific name of a library or function) so that the function pointers for that cache system will be initialized and the linker will pull in only the desired cache library. Below is an example of `cacheXxxLib` being linked in by `sysLib.c`. For systems without caches and for those architectures with only one cache design, there is no need for the `sysCacheLibInit` variable.

```
FUNCPTR sysCacheLibInit = (FUNCPTR) cacheXxxLibInit;
```

For cache systems with bus snooping, the flush and invalidate macros should be NULLified to enhance system and driver performance in `sysHwInit()`.

```
void sysHwInit ()
{
...
cacheLib.flushRtn = NULL;          /* no flush necessary */
cacheLib.invalidateRtn = NULL;    /* no invalidate necessary */
...
}
```

There may be some drivers that require numerous cache calls, so many that they interfere with the code clarity. Additional checking can be done at the initialization stage to determine if `cacheDmaMalloc()` returned a buffer in non-cacheable space. Remember that it will return a cache-safe buffer by virtue of the function pointers. Ideally, these are NULL, since the MMU was used to mark the pages as non-cacheable. The macros `CACHE_Xxx_IS_WRITE_COHERENT` and `CACHE_Xxx_IS_READ_COHERENT` can be used to check the flush and invalidate function pointers, respectively.

Write buffers are used to allow the processor to continue execution while the bus interface unit moves the data to the external device. In theory, the write buffer should be smart enough to flush itself when there is a write to non-cacheable space or a read of an item that is in the buffer. In those cases where the hardware does not support this, the software must flush the buffer manually. This often is accomplished by a read to non-cacheable space or a NOP instruction that serializes the chip's pipelines and buffers. This is not really a caching issue; however, the cache library provides a `CACHE_PIPE_FLUSH` macro. External write buffers may still need to be handled in a board-specific manner.

| | |
|----------------------|---|
| INCLUDE FILES | cacheLib.h |
| SEE ALSO | Architecture-specific cache-management libraries (cacheXxxLib), vmLib , <i>VxWorks Programmer's Guide: I/O System</i> |

cacheMb930Lib

| | |
|----------------------|--|
| NAME | cacheMb930Lib – Fujitsu MB86930 (SPARClite) cache management library |
| ROUTINES | <i>cacheMb930LibInit()</i> – initialize the Fujitsu MB86930 cache library <i>cacheMb930LockAuto()</i> – enable MB86930 automatic locking of kernel instructions/data <i>cacheMb930ClearLine()</i> – clear a line from an MB86930 cache |
| DESCRIPTION | This library contains architecture-specific cache library functions for the Fujitsu MB86930 (SPARClite) architecture. There are separate small instruction and data caches on chip, both of which operate in write-through mode. Each cache line contains 16 bytes. Cache tags may be "flushed" by accesses to alternate space in supervisor mode. Invalidate operations are performed in software by writing zero to the cache tags in an iterative manner. Locked data cache tags are not invalidated since the data resides only in the cache and not in RAM. The global and local cache locking features are beneficial for real-time systems. Note that there is no MMU (Memory Management Unit) support. For general information about caching, see the manual entry for cacheLib . |
| INCLUDE FILES | arch/sparc/sparclite.h , cacheLib.h |
| SEE ALSO | cacheLib |

cacheMicroSparcLib

| | |
|--------------------|--|
| NAME | cacheMicroSparcLib – microSPARC cache management library |
| ROUTINES | <i>cacheMicroSparcLibInit()</i> – initialize the microSPARC cache library |
| DESCRIPTION | This library contains architecture-specific cache library functions for the microSPARC architecture. Currently two microSPARC CPU are supported: the Texas Instrument TMS3900S10 (also known as Tsunami) and the FUJITSU MB86904 (also know as Swift). The TMS390S10 implements a 4-Kbyte Instruction and a 2-Kbyte Data cache, the MB86904 a 16-Kbyte Instruction and a 8-Kbyte Data cache. Both operate in write-through mode. |

The Instruction Cache Line size is 32 bytes while the Data Cache Line size is 16 bytes, but for memory allocation purposes, a cache line alignment size of 32 bytes will be assumed. The TMS390S10 either cache only supports invalidation of all entries and no cache locking is available, the MB86904 supports a per cache line invalidation, with specific alternate stores, but no cache locking

MMU (Memory Management Unit) support is needed to mark pages cacheable or non-cacheable. For more information, see the manual entry for **vmLib**.

For general information about caching, see the manual entry for **cacheLib**.

INCLUDE FILES **cacheLib.h**

SEE ALSO **cacheLib**, **vmLib**

cacheR3kALib

NAME **cacheR3kALib** – MIPS R3000 cache management assembly routines

ROUTINES *cacheR3kDsize()* – return the size of the R3000 data cache
cacheR3kIsize() – return the size of the R3000 instruction cache

DESCRIPTION This library contains MIPS R3000 cache set-up and invalidation routines written in assembly language. The R3000 utilizes a variable-size instruction and data cache that operates in write-through mode. Cache line size also varies. Cache tags may be invalidated on a per-word basis by execution of a byte write to a specified word while the cache is isolated. See also the manual entry for **cacheR3kLib**.

For general information about caching, see the manual entry for **cacheLib**.

INCLUDE FILES **cacheLib.h**

SEE ALSO **cacheR3kLib**, **cacheLib**, Gerry Kane: *MIPS R3000 RISC Architecture*

cacheR3kLib

NAME **cacheR3kLib** – MIPS R3000 cache management library

ROUTINES *cacheR3kLibInit()* – initialize the R3000 cache library

| | |
|----------------------|---|
| DESCRIPTION | This library contains architecture-specific cache library functions for the MIPS R3000 architecture. The R3000 utilizes a variable-size instruction and data cache that operates in write-through mode. Cache line size also varies. Cache tags may be invalidated on a per-word basis by execution of a byte write to a specified word while the cache is isolated. See also the manual entry for cacheR3kALib . For general information about caching, see the manual entry for cacheLib . |
| INCLUDE FILES | cacheLib.h |
| SEE ALSO | cacheR3kALib , cacheLib , Gerry Kane: <i>MIPS R3000 RISC Architecture</i> |

cacheR4kLib

| | |
|----------------------|--|
| NAME | cacheR4kLib – MIPS R4000 cache management library |
| ROUTINES | <i>cacheR4kLibInit()</i> – initialize the R4000 cache library |
| DESCRIPTION | This library contains architecture-specific cache library functions for the MIPS R4000 architecture. The R4000 utilizes a variable-size instruction and data cache that operates in write-back mode. Cache line size also varies. For general information about caching, see the manual entry for cacheLib . |
| INCLUDE FILES | cacheLib.h |
| SEE ALSO | cacheLib |

cacheR33kLib

| | |
|--------------------|--|
| NAME | cacheR33kLib – MIPS R33000 cache management library |
| ROUTINES | <i>cacheR33kLibInit()</i> – initialize the R33000 cache library |
| DESCRIPTION | This library contains architecture-specific cache library functions for the MIPS R33000 architecture. The R33000 utilizes a 8-Kbyte instruction cache and a 1-Kbyte data cache that operate in write-through mode. Cache line size is fixed at 16 bytes. Cache tags may be invalidated on a per-line basis by execution of a store to a specified line while the cache is in invalidate mode. For general information about caching, see the manual entry for cacheLib . |

INCLUDE FILES `arch/mips/lr33000.h, cacheLib.h`

SEE ALSO `cacheLib`, *LSI Logic LR33000 MIPS Embedded Processor User's Manual*

cacheR333x0Lib

NAME `cacheR333x0Lib` – MIPS R333x0 cache management library

ROUTINES `cacheR333x0LibInit()` – initialize the R333x0 cache library

DESCRIPTION This library contains architecture-specific cache library functions for the MIPS R333x0 architecture. The R33300 utilizes a 4-Kbyte instruction cache and a 2-Kbyte data cache that operate in write-through mode. The R33310 utilizes a 8-Kbyte instruction cache and a 4-Kbyte data cache that operate in write-through mode. Cache line size is fixed at 16 bytes. Cache tags may be invalidated on a per-line basis by execution of a store to a specified line while the cache is in invalidate mode.

For general information about caching, see the manual entry for `cacheLib`.

INCLUDE FILES `arch/mips/lr33300.h, cacheLib.h`

SEE ALSO `cacheLib`, *LSI Logic LR33300 and LR33310 Self-Embedding Processors User's Manual*

cacheSun4Lib

NAME `cacheSun4Lib` – Sun-4 cache management library

ROUTINES `cacheSun4LibInit()` – initialize the Sun-4 cache library
`cacheSun4ClearLine()` – clear a line from a Sun-4 cache
`cacheSun4ClearPage()` – clear a page from a Sun-4 cache
`cacheSun4ClearSegment()` – clear a segment from a Sun-4 cache
`cacheSun4ClearContext()` – clear a specific context from a Sun-4 cache

DESCRIPTION This library contains architecture-specific cache library functions for the Sun Microsystems Sun-4 architecture. There is a 64-Kbyte mixed instruction and data cache that operates in write-through mode. Each cache line contains 16 bytes. Cache tags may be "flushed" by accesses to alternate space in supervisor mode. Invalidate operations are performed in software by writing zero to the cache tags in an iterative manner. Tag operations are performed on "page," "segment," or "context" granularity.

MMU (Memory Management Unit) support is needed to mark pages cacheable or non-cacheable. For more information, see the manual entry for **vmLib**.

For general information about caching, see the manual entry for **cacheLib**.

INCLUDE FILES **cacheLib.h**

SEE ALSO **cacheLib, vmLib**

cacheTiTms390Lib

NAME **cacheTiTms390Lib** – TI TMS390 SuperSPARC cache management library

ROUTINES *cacheTiTms390LibInit()* – initialize the TI TMS390 cache library
cacheTiTms390VirtToPhys() – translate a virtual address for **cacheLib**
cacheTiTms390PhysToVirt() – translate a physical address for drivers
cleanUpStoreBuffer() – clean up store buffer after a data store error interrupt

DESCRIPTION This library contains architecture-specific cache library functions for the TI TMS390 SuperSPARC architecture. The on-chip cache architecture is explained in the first table below. Note, the data cache mode depends on whether there is an external Multicache Controller (MCC). Both on-chip caches support cache coherency via snooping and line locking. For memory allocation purposes, a cache line alignment size of 64 bytes is assumed. The MCC supports cache coherency via snooping, but does not support line locking.

| Cache Type | Size | Lines | Sets | Ways | Line Size (Bytes) | Mode |
|------------|------|-------|------|------|-------------------|--|
| Instr | 20K | 320 | 64 | 5 | 2*32 | never written back |
| Data | 16K | 512 | 128 | 4 | 32 | with MCC: Write-through without MCC: Copy-back with write allocation |

The cache operations provided are explained in the table below. Operations marked "Hardware" and "Software" are implemented as marked, and are fast and slow, respectively. Operations marked "NOP" return OK without doing anything. Operations with another operation name perform that operation rather than their own. Partial operations marked "Entire" actually perform an "Entire" operation. When the MCC is installed, operations upon the data cache are performed upon both the data cache and the MCC. Lines "Data-Data" and "Data-MCC" describe the data cache and MCC, respectively, portions of a data cache operation.

| MCC: | | No | No | Yes | Yes | Yes |
|--------------------------|---------|--------|-------|--------|-----------|----------|
| Cache Type: | | Instr | Data | Instr | Data-Data | Data-MCC |
| <i>cacheInvalidate()</i> | entire | H/W | H/W | H/W | H/W | S/W |
| | partial | Entire | S/W | Entire | S/W | S/W |
| <i>cacheFlush()</i> | entire | NOP | Clear | NOP | NOP | S/W |
| | partial | NOP | Clear | NOP | NOP | Clear |
| <i>cacheClear()</i> | entire | H/W | S/W | H/W | H/W | S/W |
| | partial | Entire | S/W | Entire | S/W | S/W |
| <i>cacheLock()</i> and | entire | S/W | S/W | S/W | S/W | NOP |
| <i>cacheUnlock()</i> | partial | S/W | S/W | S/W | S/W | NOP |

The architecture of the optional Multicache Controller (MCC) is explained in the table below. The MCC supports cache coherency via snooping, and does not support line locking.

The MCC does not have a `CACHE_TYPE` value for *cacheEnable()* or *cacheDisable()*. For enable and disable operations, the MCC is treated as an extension of both the on-chip data and instruction caches. If either the data or instruction caches are enabled, the MCC is enabled. If both the data and the instruction caches are disabled, the MCC is disabled. For invalidate, flush, and clear operations the MCC is treated as an extension of only the on-chip data cache. The *cacheInvalidate()*, *cacheFlush()*, and *cacheClear()* operations for the instruction cache operate only on the on-chip instruction cache. However these operations for the data cache operate on both the on-chip data cache and the MCC.

| Cache Type | Size | Blocks | Ways | Block Size (bytes) | Mode |
|-------------|--------------|------------|------|--------------------|-----------|
| MCC on MBus | 0, 1M | 0, 8K | 1 | 4*32 | Copy-back |
| MCC on XBus | 512K, 1M, 2M | 2K, 4K, 8K | 1 | 4*64 | Copy-back |

Any input peripheral that does not support cache coherency may be accessed through either a cached buffer with a partial *cacheTiTms390Invalidate()* operation, or an uncached buffer without it. (*cacheInvalidate()* cannot be used; it is a NOP since it assumes cache coherency.) Choose whichever is faster for the application.

Any output peripheral that does not support cache coherency may be accessed through either a cached buffer with a partial *cacheTiTms390Flush()* operation, or an uncached buffer without it. (*cacheFlush()* cannot be used; it is a NOP since it assumes cache coherency.) Choose whichever is faster for the application.

Any peripheral that supports cache coherency should be accessed through a cached buffer without using any of the above operations. Using either an uncached buffer or any of the above operations will just slow the system down.

MMU (Memory Management Unit) support is needed to mark pages cacheable or non-cacheable. For more information, see the manual entry for **vmLib**.

For general information about caching, see the manual entry for **cacheLib**.

INCLUDE FILES `cacheLib.h`

SEE ALSO `cacheLib`, `vmLib`

cd2400Sio

NAME `cd2400Sio` – CL-CD2400 MPCC serial driver

ROUTINES `cd2400HrdInit()` – initialize the chip
`cd2400IntRx()` – handle receiver interrupts
`cd2400IntTx()` – handle transmitter interrupts
`cd2400Int()` – handle special status interrupts

DESCRIPTION This is the driver for the Cirrus Logic CD2400 MPCC. It uses the SCC's in asynchronous mode.

USAGE A `CD2400_QUSART` structure is used to describe the chip. This data structure contains four `CD2400_CHAN` structure which describe the chip's four serial channels. The BSP's `sysHwInit()` routine typically calls `sysSerialHwInit()` which initializes all the values in the `CD2400_QUSART` structure (except the `SIO_DRV_FUNCS`) before calling `cd2400HrdInit()`. The BSP's `sysHwInit2()` routine typically calls `sysSerialHwInit2()` which connects the chips interrupts (`cd2400Int`, `cd2400IntRx`, and `cd2400IntTx`) via `intConnect()`.

IOCTL FUNCTIONS This driver responds to the same `ioctl()` codes as a normal serial driver; for more information, see the comments in `sioLib.h`. The available baud rates are: 50, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200, and 38400.

INCLUDE FILES `drv/sio/cd2400Sio.h`

cdromFsLib

NAME `cdromFsLib` – ISO 9660 CD-ROM read-only file system library

ROUTINES `cdromFsInit()` – initialize `cdromFsLib`
`cdromFsVolConfigShow()` – show the volume configuration information
`cdromFsDevCreate()` – create a `cdromFsLib` device

DESCRIPTION

This library defines **cdromFsLib**, a utility that lets you use standard POSIX I/O calls to read data from a CD-ROM formatted according to the ISO 9660 standard file system.

It provides access to CD-ROM file systems using any standard **BLOCK_DEV** structure (that is, a disk-type driver).

The basic initialization sequence is similar to installing a DOS file system on a SCSI device.

1. Initialize the cdrom file system library (preferably in *sysScsiConfig()* in **sysScsi.c**):

```
cdromFsInit ();
```

2. Locate and create a SCSI physical device:

```
pPhysDev=scsiPhysDevCreate(pSysScsiCtrl,0,0,0,NONE,1,0,0);
```

3. Create a SCSI block device on the physical device:

```
pBlkDev = (SCSI_BLK_DEV *) scsiBlkDevCreate (pPhysDev, 0, 0);
```

4. Create a CD-ROM file system on the block device:

```
cdVolDesc = cdromFsDevCreate ("cdrom:", (BLK_DEV *) pBlkDev);
```

Call *cdromFsDevCreate()* once for each CD-ROM drive attached to your target. After the successful completion of *cdromFsDevCreate()*, the CD-ROM file system will be available like any DOS file system, and you can access data on the named CD-ROM device using *open()*, *close()*, *read()*, *ioctl()*, *readdir()*, and *stat()*. A *write()* always returns an error.

The **cdromFsLib** utility supports multiple drives, concurrent access from multiple tasks, and multiple open files.

FILE AND DIRECTORY NAMING

The strict ISO 9660 specification allows only uppercase file names consisting of 8 characters plus a 3 character suffix. To support multiple versions of the same file, the ISO 9660 specification also supports version numbers. When specifying a file name in an *open()* call, you can select the file version by appending the file name with a semicolon (;) followed by a decimal number indicating the file version. If you omit the version number, **cdromFsLib** opens the latest version of the file.

To accommodate users familiar with MS-DOS, **cdromFsLib** lets you use lowercase name arguments to access files with names consisting entirely of uppercase characters. Mixed-case file and directory names are accessible only if you specify their exact case-correct names.

For the time being, **cdromFsLib** further accommodates MS-DOS users by allowing "stead of "/" in pathnames. However, the use of the backslash is discouraged because it may not be supported in future versions of **cdromFsLib**.

Finally, **cdromFsLib** uses an 8-bit clean implementation of ISO 9660. Thus, **cdromFsLib** is compatible with CD-ROMs using either Latin or Asian characters in the file names.

IOCTL CODES SUPPORTED

FIOGETNAME

Returns the file name for a specific file descriptor.

FIOLABELGET

Retrieves the volume label. This code can be used to verify that a particular volume has been inserted into the drive.

FIOWHERE

Determines the current file position.

FIOSEEK

Changes the current file position.

FIONREAD

Tells you the number of bytes between the current location and the end of this file.

FIOREADDIR

Reads the next directory entry.

FIODISKCHANGE

Announces that a disk has been replaced (in case the block driver is not able to provide this indication).

FIOUNMOUNT

Announces that the a disk has been removed (all currently open file descriptors are invalidated).

FIOFSTATGET

Gets the file status information (directory entry data).

MODIFYING A BSP TO USE CDROMFS

The following example describes mounting cdromFS on a SCSI device.

Edit your BSP's **config.h** to make the following changes:

1. Insert the following macro definition:

```
#define INCLUDE_CDROMFS
```

2. Change FALSE to TRUE in the section under the following comment:

```
/* change FALSE to TRUE for SCSI interface */
```

Make the following changes in **sysScsi.c** (or **sysLib.c** if your BSP has no **sysScsi.c**):

1. Add the following declaration to the top of the file:

```
#ifndef INCLUDE_CDROMFS  
#include "cdromFsLib.h"  
STATUS cdromFsInit (void);  
#endif
```

2. Modify the definition of **sysScsiInit()** to include the following:

```
#ifdef INCLUDE_CDROMFS
cdromFsInit();
#endif
```

The call to *cdromFsInit()* initializes cdromFS. This call must be made only once and must complete successfully before you can call any other *cdromFsLib* routines, such as *cdromFsDevCreate()*. Typically, you make the *cdromFsInit()* call at system startup. Because cdromFS is used with SCSI CD-ROM devices, it is natural to call *cdromFsInit()* from within *sysScsiInit()*.

3. Modify the definition of *sysScsiConfig()* (if included in your BSP) to include the following:

```
/* configure a SCSI CDROM at busId 6, LUN = 0 */
#ifdef INCLUDE_CDROMFS
if ((pSpd60 = scsiPhysDevCreate (pSysScsiCtrl, 6, 0, 0, NONE, 0, 0, 0)) ==
    (SCSI_PHYS_DEV *) NULL)
    {
        SCSI_DEBUG_MSG ("sysScsiConfig: scsiPhysDevCreate failed for CDROM.\n",
            0, 0, 0, 0, 0, 0);
        return (ERROR);
    }
else if ((pSbdCd = scsiBlkDevCreate (pSpd60, 0, 0) ) == NULL)
    {
        SCSI_DEBUG_MSG ("sysScsiConfig: scsiBlkDevCreate failed for CDROM.\n",
            0, 0, 0, 0, 0, 0);
        return (ERROR);
    }
/*
 * Create an instance of a CD-ROM device in the I/O system.
 * A block device must already have been created. Internally,
 * cdromFsDevCreate() calls iosDrvInstall(), which enters the
 * appropriate driver routines in the I/O driver table.
 */
if ((cdVolDesc = cdromFsDevCreate ("cdrom:", (BLK_DEV *) pSbdCd )) == NULL)
    {
        return (ERROR);
    }
#endif /* end of #ifdef INCLUDE_CDROMFS */
```

4. Before the definition of *sysScsiConfig()*, declare the following global variables used in the above code fragment:

```
SCSI_PHYS_DEV *pSpd60;
BLK_DEV *pSbdCd;
CDROM_VOL_DESC_ID cdVolDesc;
```

The main goal of the above code fragment is to call *cdromFsDevCreate()*. As input, *cdromFsDevCreate()* expects a pointer to a block device. In the example above, the

cisLib

scsiPhysDevCreate() and *scsiBlkDevCreate()* calls set up a block device interface for a SCSI CD-ROM device.

After the successful completion of *cdromFsDevCreate()*, the device called "cdrom" is accessible using the standard *open()*, *close()*, *read()*, *ioctl()*, *readdir()*, and *stat()* calls.

INCLUDE FILES **cdromFsLib.h**

CAVEATS The **cdromFsLib** utility does not support CD sets containing multiple disks.

SEE ALSO **ioLib**, ISO 9660 Specification

cisLib

NAME **cisLib** – PCMCIA CIS library

ROUTINES *cisGet()* – get information from a PC card's CIS
cisFree() – free tuples from the linked list
cisConfigregGet() – get the PCMCIA configuration register
cisConfigregSet() – set the PCMCIA configuration register

DESCRIPTION This library contains routines to manipulate the CIS (Configuration Information Structure) tuples and the card configuration registers. The library uses a memory window which is defined in **pcmciaMemwint** to access the CIS of a PC card. All CIS tuples in a PC card are read and stored in a linked list, **cisTupleList**. If there are configuration tuples, they are interpreted and stored in another link list, **cisConfigList**. After the CIS is read, the PC card's enabler routine allocates resources and initializes a device driver for the PC card.

If a PC card is inserted, the CSC (Card Status Change) interrupt handler gets a CSC event from the PCMCIA chip and adds a *cisGet()* job to the PCMCIA daemon. The PCMCIA daemon initiates the *cisGet()* work. The CIS library reads the CIS from the PC card and makes a linked list of CIS tuples. It then enables the card.

If the PC card is removed, the CSC interrupt handler gets a CSC event from the PCMCIA chip and adds a *cisFree()* job to the PCMCIA daemon. The PCMCIA daemon initiates the *cisFree()* work. The CIS library frees allocated memory for the linked list of CIS tuples.

cisShow

| | |
|--------------------|--|
| NAME | cisShow – PCMCIA CIS show library |
| ROUTINES | <i>cisShow()</i> – show CIS information |
| DESCRIPTION | This library provides a show routine for CIS tuples. |

clockLib

| | |
|-----------------------|--|
| NAME | clockLib – clock library (POSIX) |
| ROUTINES | <i>clock_getres()</i> – get the clock resolution (POSIX) <i>clock_setres()</i> – set the clock resolution <i>clock_gettime()</i> – get the current time of the clock (POSIX) <i>clock_settime()</i> – set the clock to a specified time (POSIX) |
| DESCRIPTION | <p>This library provides a clock interface, as defined in the IEEE standard, POSIX 1003.1b.</p> <p>A clock is a software construct that keeps time in seconds and nanoseconds. The clock has a simple interface with three routines: <i>clock_settime()</i>, <i>clock_gettime()</i>, and <i>clock_getres()</i>. The non-POSIX routine <i>clock_setres()</i> is provided (temporarily) so that clockLib is informed if there are changes in the system clock rate (e.g., after a call to <i>sysClkRateSet()</i>).</p> <p>Times used in these routines are stored in the timespec structure:</p> <pre>struct timespec { time_t tv_sec; /* seconds */ long tv_nsec; /* nanoseconds (0 -1,000,000,000) */ };</pre> |
| IMPLEMENTATION | Only one <i>clock_id</i> is supported, the required CLOCK_REALTIME . Conceivably, additional "virtual" clocks could be supported, or support for additional auxiliary clock hardware (if available) could be added. |
| INCLUDE FILES | timers.h |
| SEE ALSO | IEEE <i>VxWorks Programmer's Guide: Basic OS</i> , POSIX 1003.1b documentation |

cplusLib

| | |
|--------------------|--|
| NAME | cplusLib – basic run-time support for C++ |
| ROUTINES | <i>cplusCallNewHandler()</i> – call the allocation failure handler (C++) <i>cplusCtors()</i> – call static constructors (C++) <i>cplusCtorsLink()</i> – call all linked static constructors (C++) <i>cplusDemanglerSet()</i> – change C++ demangling mode (C++) <i>cplusDtors()</i> – call static destructors (C++) <i>cplusDtorsLink()</i> – call all linked static destructors (C++) <i>cplusLibInit()</i> – initialize the C++ library (C++) <i>cplusXtorSet()</i> – change C++ static constructor calling strategy (C++) <i>operator delete()</i> – default run-time support for memory deallocation (C++) <i>operator new()</i> – default run-time support for operator new (C++) <i>operator new()</i> – default run-time support for operator new (nothrow) (C++) <i>operator new()</i> – run-time support for operator new with placement (C++) <i>set_new_handler()</i> – set new_handler to user-defined function (C++) <i>set_terminate()</i> – set terminate to user-defined function (C++) |
| DESCRIPTION | <p>This library provides run-time support and shell utilities that support the development of VxWorks applications in C++. The run-time support can be broken into three categories:</p> <ul style="list-style-type: none">– Support for C++ new and delete operators.– Support for initialization and cleanup of static objects. <p>Shell utilities are provided for:</p> <ul style="list-style-type: none">– Resolving overloaded C++ function names.– Hiding C++ name mangling, with support for terse or complete name demangling.– Manual or automatic invocation of static constructors and destructors. <p>The usage of cplusLib is more fully described in the <i>VxWorks Programmer's Guide: C++ Development</i>.</p> |
| SEE ALSO | <i>VxWorks Programmer's Guide: C++ Development</i> |

dbgArchLib

| | | |
|-----------------|--|--|
| NAME | dbgArchLib – architecture-dependent debugger library | |
| ROUTINES | <p> <i>g0()</i> – return the contents of register g0, also g1 – g7 (SPARC) and g1 – g14 (i960) <i>a0()</i> – return the contents of register a0 (also a1 – a7) (MC680x0) <i>d0()</i> – return the contents of register d0 (also d1 – d7) (MC680x0) <i>sr()</i> – return the contents of the status register (MC680x0) <i>psrShow()</i> – display the meaning of a specified psr value, symbolically (SPARC) <i>fsrShow()</i> – display the meaning of a specified fsr value, symbolically (SPARC) <i>o0()</i> – return the contents of register o0 (also o1 – o7) (SPARC) <i>l0()</i> – return the contents of register l0 (also l1 – l7) (SPARC) <i>i0()</i> – return the contents of register i0 (also i1 – i7) (SPARC) <i>npc()</i> – return the contents of the next program counter (SPARC) <i>psr()</i> – return the contents of the processor status register (SPARC) <i>wim()</i> – return the contents of the window invalid mask register (SPARC) <i>y()</i> – return the contents of the y register (SPARC) <i>pfpr()</i> – return the contents of register pfpr (i960) <i>tsp()</i> – return the contents of register sp (i960) <i>rip()</i> – return the contents of register rip (i960) <i>r3()</i> – return the contents of register r3 (also r4 – r15) (i960) <i>fp()</i> – return the contents of register fp (i960) <i>fp0()</i> – return the contents of register fp0 (also fp1 – fp3) (i960KB, i960SB) <i>pcw()</i> – return the contents of the pcw register (i960) <i>tcw()</i> – return the contents of the tcw register (i960) <i>acw()</i> – return the contents of the acw register (i960) <i>dbgBpTypeBind()</i> – bind a breakpoint handler to a breakpoint type (MIPS R3000, R4000) <i>edi()</i> – return the contents of register edi (also esi – eax) (i386/i486) <i>eflags()</i> – return the contents of the status register (i386/i486) <i>r0()</i> – return the contents of register r0 (also r1 – r14) (ARM) <i>cpsr()</i> – return the contents of the current processor status register (ARM) <i>psrShow;1()</i> – display the meaning of a specified PSR value, symbolically (ARM) </p> | |

DESCRIPTION This module provides architecture-specific support functions for **dbgLib**. It also includes user-callable functions for accessing the contents of registers in a task's TCB (task control block). These routines include:

| | | |
|----------|---------------------------|--|
| MC680x0: | <i>a0()</i> – <i>a7()</i> | – address registers (a0 – a7) |
| | <i>d0()</i> – <i>d7()</i> | – data registers (d0 – d7) |
| | <i>sr()</i> | – status register (sr) |
| SPARC | <i>psrShow()</i> | – psr value, symbolically |
| | <i>fsrShow()</i> | – fsr value, symbolically |
| | <i>g0()</i> – <i>g7()</i> | – global registers (g0 – g7) |

| | | |
|------------|-----------------------------|--|
| | <i>o0()</i> – <i>o7()</i> | – out registers (o0 – o7 , note lower-case "o") |
| | <i>l0()</i> – <i>l7()</i> | – local registers (l0 – l7 , note lower-case "l") |
| | <i>i0()</i> – <i>i7()</i> | – in registers (i0 – i7) |
| | <i>npc()</i> | – next program counter (npc) |
| | <i>psr()</i> | – processor status register (psr) |
| | <i>wim()</i> | – window invalid mask (wim) |
| | <i>y()</i> | – y register |
| i960: | <i>g0()</i> – <i>g14()</i> | – global registers |
| | <i>r3()</i> – <i>r15()</i> | – local registers |
| | <i>tsp()</i> | – stack pointer |
| | <i>rip()</i> | – return instruction pointer |
| | <i>pfp()</i> | – previous frame pointer |
| | <i>fp()</i> | – frame pointer |
| | <i>fp0()</i> – <i>fp3()</i> | – floating-point registers (i960 KB and SB only) |
| | <i>pcw()</i> | – processor control word |
| | <i>tcw()</i> | – trace control word |
| | <i>acw()</i> | – arithmetic control word |
| MIPS | <i>dbgBpTypeBind()</i> | – bind a breakpoint handler to a breakpoint type |
| i386/i486: | <i>edi()</i> – <i>eax()</i> | – named register values |
| | <i>eflags()</i> | – status register value |
| ARM | <i>r0()</i> – <i>r14()</i> | – general-purpose registers (r0 – r14) |
| | <i>cpsr()</i> | – current processor status reg (cpsr) |
| | <i>psrShow()</i> | – psr value, symbolically |

Note: The routine *pc()*, for accessing the program counter, is found in **usrLib**.

SEE ALSO

dbgLib, *VxWorks Programmer's Guide: Target Shell*

dbgLib

NAME

dbgLib – debugging facilities

ROUTINES

dbgHelp() – display debugging help menu
dbgInit() – initialize the local debugging package
b() – set or display breakpoints
e() – set or display eventpoints (WindView)
bh() – set a hardware breakpoint
bd() – delete a breakpoint
bdall() – delete all breakpoints
c() – continue from a breakpoint

cret() – continue until the current subroutine returns
s() – single-step a task
so() – single-step, but step over a subroutine
l() – disassemble and display a specified number of instructions
tt() – display a stack trace of a task

DESCRIPTION

This library contains VxWorks’s primary interactive debugging routines, which provide the following facilities:

- task breakpoints
- task single-stepping
- symbolic disassembly
- symbolic task stack tracing

In addition, **dbgLib** provides the facilities necessary for enhanced use of other VxWorks functions, including:

- enhanced shell abort and exception handling (via **tyLib** and **exLib**)

The facilities of **exLib** are used by **dbgLib** to support breakpoints, single-stepping, and additional exception handling functions.

INITIALIZATION

The debugging facilities provided by this module are optional. In the standard VxWorks development configuration as distributed, the debugging package is included. The configuration macro is **INCLUDE_DEBUG**. When defined, it enables the call to *dbgInit()* in the task *usrRoot()* in **usrConfig.c**. The *dbgInit()* routine initializes **dbgLib** and must be made before any other routines in the module are called.

BREAKPOINTS

Use the routine *b()* or *bh()* to set breakpoints. Breakpoints can be set to be hit by a specific task or all tasks. Multiple breakpoints for different tasks can be set at the same address. Clear breakpoints with *bd()* and *bdall()*.

When a task hits a breakpoint, the task is suspended and a message is displayed on the console. At this point, the task can be examined, traced, deleted, its variables changed, etc. If you examine the task at this point (using the *i()* routine), you will see that it is in a suspended state. The instruction at the breakpoint address has not yet been executed.

To continue executing the task, use the *c()* routine. The breakpoint remains until it is explicitly removed.

EVENTPOINTS (WINDVIEW)

When WindView is installed, **dbgLib** supports eventpoints. Use the routine *e()* to set eventpoints. Eventpoints can be set to be hit by a specific task or all tasks. Multiple eventpoints for different tasks can be set at the same address.

When a task hits an eventpoint, an event is logged and is displayed by VxWorks kernel instrumentation.

You can manage eventpoints with the same facilities that manage breakpoints: for example, unbreakable tasks (discussed below) ignore eventpoints, and the **b()** command (without arguments) displays eventpoints as well as breakpoints. As with breakpoints, you can clear eventpoints with **bd()** and **bdall()**.

UNBREAKABLE TASKS

An *unbreakable* task ignores all breakpoints. Tasks can be spawned unbreakable by specifying the task option **VX_UNBREAKABLE**. Tasks can subsequently be set unbreakable or breakable by resetting **VX_UNBREAKABLE** with *taskOptionsSet()*. Several VxWorks tasks are spawned unbreakable, such as the shell, the exception support task *excTask()*, and several network-related tasks.

DISASSEMBLER AND STACK TRACER

The *I()* routine provides a symbolic disassembler. The *tt()* routine provides a symbolic stack tracer.

SHELL ABORT AND EXCEPTION HANDLING

This package includes enhanced support for the shell in a debugging environment. The terminal abort function, which restarts the shell, is invoked with the abort key if the **OPT_ABORT** option has been set. By default, the abort key is CTRL-C. For more information, see the manual entries for *tyAbortSet()* and *tyAbortFuncSet()*.

THE DEFAULT TASK AND TASK REFERENCING

Many routines in this module take an optional task name or ID as an argument. If this argument is omitted or zero, the "current" task is used. The current task (or "default" task) is the last task referenced. The **dbgLib** library uses *taskIdDefault()* to set and get the last-referenced task ID, as do many other VxWorks routines.

All VxWorks shell expressions can reference a task by either ID or name. The shell attempts to resolve a task argument to a task ID; if no match is found in the system symbol table, it searches for the argument in the list of active tasks. When it finds a match, it substitutes the task name with its matching task ID. In symbol lookup, symbol names take precedence over task names.

CAVEAT

When a task is continued, *c()* and *s()* routines do not yet distinguish between a suspended task or a task suspended by the debugger. Therefore, use of these routines should be restricted to only those tasks being debugged.

INCLUDE FILES **dbgLib.h**

SEE ALSO **dbgArchLib**, **excLib**, **tyLib**, *taskIdDefault()*, *taskOptionsSet()*, *tyAbortSet()*, *tyAbortFuncSet()*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

dec21x4xEnd

| | |
|---------------------------|--|
| NAME | dec21x4xEnd – END style DEC 21x4x PCI Ethernet network interface driver |
| ROUTINES | <i>dec21x4xEndLoad()</i> – initialize the driver and device |
| DESCRIPTION | <p>This module implements a DEC 21x4x PCI Ethernet network interface driver and supports 21040, 21140 and 21143 versions of the chip.</p> <p>The DEC 21x4x PCI Ethernet controller is little endian because it interfaces with a little endian PCI bus. Although PCI configuration for a device is handled in the BSP, all other device programming and initialization are handled in this module.</p> <p>This driver is designed to be moderately generic. Without modification, it can operate across the range of architectures and targets supported by VxWorks. To achieve this, the driver requires a few external support routines as well as several target-specific parameters. These parameters, and the mechanisms used to communicate them to the driver, are detailed below. If any of the assumptions stated below are not true for your particular hardware, you need to modify the driver before it can operate correctly on your hardware.</p> <p>On 21040, the driver configures the 10BASE-T interface by default, waits for two seconds, and checks the status of the link. If the link status indicates failure, AUI interface is configured.</p> <p>On other versions of the 2114x family, the driver reads media information from a DEC serial ROM and configures the media. On targets that do not support a DEC format serial ROM, the driver calls a target-specific media select routine using the hook, <i>_func_dec2114xMediaSelect</i>, to configure the media.</p> <p>The driver supports big-endian or little-endian architectures (as a configurable option). The driver also contains error recovery code that handles known device errata related to DMA activity.</p> <p>Big endian processors can be connected to the PCI bus through some controllers which take care of hardware byte swapping. In such cases all the registers which the chip DMAs to have to be swapped and written to, so that when the hardware swaps the accesses, the chip would see them correctly. The chip still has to be programmed to operate in little endian mode as it is on the PCI bus. If the cpu board hardware automatically swaps all the accesses to and from the PCI bus, then input and output byte stream need not be swapped.</p> |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | The driver provides one standard external interface, <i>dec21x4xEndLoad()</i> , which takes a string of colon separated parameters. The parameters should be specified as hexadecimal |

strings, optionally preceded by "0x" or a minus sign "-".

Although the parameter string is parsed using `strtok_r()`, each parameter is converted from string to binary by a call to `strtoul(parameter, NULL, 16)`.

The format of the parameter string is:

"unit number:device addr:PCI addr:ivec:ilevel:mem base: mem size:user flags:offset"

TARGET-SPECIFIC PARAMETERS

unit number

This represents the device instance number relative to this driver. I.e. a value of zero represents the first dec21x4x device, a value of 1 represents the second dec21x4x device.

device addr

This is the base address at which the hardware device registers are located.

PCI addr

This parameter defines the main memory address over the PCI bus. It is used to translate physical memory address into PCI accessible address.

ivec

This is the interrupt vector number of the hardware interrupt generated by this Ethernet device. The driver uses `intConnect`, or `pciIntConnect` (x86 arch), to attach an interrupt handler for this interrupt.

ilevel

This parameter defines the level of the hardware interrupt.

mem base

This parameter specifies the base address of a DMA-able, cache free, pre-allocated memory region for use as a memory pool for transmit/receive descriptors and buffers.

If there is no pre-allocated memory available for the driver, this parameter should be -1 (NONE). In which case, the driver allocates cache safe memory for its use using `cacheDmaAlloc()`.

mem size

The memory size parameter specifies the size of the pre-allocated memory region. If memory base is specified as NONE (-1), the driver ignores this parameter.

user flags

User flags control the run-time characteristics of the Ethernet chip. Most flags specify non default CSR0 bit values. Refer to `dec21x4xEnd.h` for the bit values of the flags, and to the device hardware reference manual for details about device capabilities, and CSR 0.

Some of them are worth mentioning:

Full Duplex Mode: When set, the `DEC_USR_FD` flag allows the device to work in full duplex mode, as long as the PHY used has this capability. It is worth noting here that in this operation mode, the dec21x40 chip ignores the Collision and the Carrier Sense signals.

Transmit treshold value: The `DEC_USR_THR_XXX` flags enable the user to choose among different threshold values for the transmit FIFO. Transmission starts when the frame size within the transmit FIFO is larger than the treshold value. This should be selected taking into account the actual operating speed of the PHY. Again, see the device hardware reference manual for details.

offset

This parameter defines the offset which is used to solve alignment problem.

Device Type

Although the default device type is DEC 21040, specifying the `DEC_USR_21140` or `DEC_USR_21143` flag bit turns on DEC 21140 or `DEC_USR_21143` functionality.

Ethernet Address

The Ethernet address is retrieved from standard serial ROM on DEC 21040, DEC 21140 and DEC 21143 devices. If retrieve from ROM fails, the driver calls the BSP routine, `sysDec21x4xEnetAddrGet()`. Specifying `DEC_USR_XEA` flag bit tells the driver should, by default, retrieve the Ethernet address using the BSP routine, `sysDec21x4xEnetAddrGet()`.

Priority RX processing

The driver programs the chip to process the transmit and receive queues at the same priority. By specifying `DEC_USR_BAR_RX`, the device is programmed to process receives at a higher priority.

TX poll rate

By default, the driver sets the Ethernet chip into a non-polling mode. In this mode, if the transmit engine is idle, it is kick-started every time a packet needs to be transmitted. Alternately, the chip can be programmed to poll for the next available transmit descriptor if the transmit engine is in idle state. The poll rate is specified by one of `DEC_USR_TAP_XXX`.

Cache Alignment

The `DEC_USR_CAL_XXX` flags specify the address boundaries for data burst transfers.

DMA burst length

The `DEC_USR_PBL_XXX` flags specify the maximum number of long words in a DMA burst.

PCI multiple read

The `DEC_USR_RML` flag specifies that a device supports PCI memory-read-multiple.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires four external support functions, and provides a hook function:

dec21x4xEnd

void sysLanIntEnable (int level)

This routine provides a target-specific interface for enabling Ethernet device interrupts at a specified interrupt level.

void sysLanIntDisable (void)

This routine provides a target-specific interface for disabling Ethernet device interrupts.

STATUS sysDec21x4xEnetAddrGet (int unit, char *enetAdrs)

This routine provides a target-specific interface for accessing a device Ethernet address.

STATUS sysDec21143Init (DRV_CTRL * pDrvCtrl)

This routine performs any target-specific initialization required before the dec21143 device is initialized by the driver. The driver calls this routine every time it wants to load the device. This routine returns OK, or ERROR if it fails.

FUNCPTR _func_dec2114xMediaSelect

This driver provides a default media select routine, when `_func_dec2114xMediaSelect` is NULL, to read and setup physical media with configuration information from a Version 3 DEC Serial ROM. Any other media configuration can be supported by initializing `<_func_dec2114xMediaSelect<`, typically in `sysHwInit()`, to a target-specific media select routine.

A media select routine is typically defined as:

```
STATUS decMediaSelect
(
    DEC21X4X_DRV_CTRL *    pDrvCtrl,    /* Driver control */
    UINT *                pCsr6Val     /* CSR6 return value */
)
{
    ...
}
```

Parameter `pDrvCtrl` is a pointer to the driver control structure which this routine may use to access the Ethernet device. The driver control structure field `mediaCount`, is initialized to 0xff at startup, while the other media control fields (`mediaDefault`, `mediaCurrent`, and `gprModeVal`) are initialized to zero. This routine may use these fields in any manner, however all other driver control fields should be considered read-only and should not be modified.

This routine should reset, initialize and select an appropriate media, and write necessary the CSR6 bits (port select, PCS, SCR, and full duplex) to memory location pointed to by `pCsr6Val`. The driver will use this value to program register CSR6. This routine should return OK, and ERROR on failure.

FUNCPTR _func_dec2114xIntAck

This driver does acknowledge the LAN interrupts. However if the board hardware requires specific interrupt acknowledgement, not provided by this driver, the BSP

should define such a routine and attach it to the driver via `_func_dec2114xIntAck`.

SEE ALSO

ifLib, *DECchip 21040 Ethernet LAN Controller for PCI. Digital Semiconductor 21140A PCI Fast Ethernet LAN Controller. Digital Semiconductor 21143 PCI/CardBus Fast Ethernet LAN Controller. Using the Digital Semiconductor 21140A with Boot ROM, Serial ROM, and External Register: An Application Note Using the Digital Semiconductor 21143 with Boot ROM, Serial ROM, and External Register: An Application Note*

dec21x40End

NAME

dec21x40End – END-style DEC 21x40 PCI Ethernet network interface driver

ROUTINES

dec21x40EndLoad() – initialize the driver and device
dec21140SromWordRead() – read two bytes from the serial ROM
dec21x40PhyLinkPoll() – Poll the PHY for link status

DESCRIPTION

This module implements a DEC 21x40 PCI Ethernet network interface driver and supports both the 21040, 21140, and 21143 versions of the chip.

The DEC 21x40 PCI Ethernet controller is little endian because it interfaces with a little-endian PCI bus. Although PCI configuration for a device is handled in the BSP, all other device programming and initialization needs are handled in this module.

This driver is designed to be moderately generic. Without modification, it can operate across the full range of architectures and targets supported by VxWorks. To achieve this, the driver requires a few external support routines as well as several target-specific parameters. These parameters, and the mechanisms used to communicate them to the driver, are detailed below. If any of the assumptions stated below are not true for your particular hardware, you need to modify the driver before it can operate correctly on your hardware.

On the 21040, the driver configures the 10BASE-T interface by default, waits for two seconds, and checks the status of the link. If the link status indicates failure, AUI interface is configured.

On other versions of the 21x40 family, the driver reads media information from a DEC serial ROM and configures the media. To configure the media on targets that do not support a DEC format serial ROM, the driver calls the target-specific media-select routine referenced in the `_func_dec21x40MediaSelect` hook.

The driver supports big-endian or little-endian architectures (as a configurable option). The driver also and contains error recovery code that handles known device errata related to DMA activity.

dec21x40End

Big-endian processors can be connected to the PCI bus through some controllers that take care of hardware byte swapping. In such cases, all the registers which the chip DMAs to have to be swapped and written to, so that when the hardware swaps the accesses, the chip would see them correctly. The chip still has to be programmed to operate in little endian mode as it is on the PCI bus. If the cpu board hardware automatically swaps all the accesses to and from the PCI bus, then input and output byte stream need not be swapped.

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

The driver provides one standard external interface, *dec21x40EndLoad()*. As input, this function expects a string of colon-separated parameters. The parameters should be specified as hexadecimal strings (optionally preceded by "0x" or a minus sign "-"). Although the parameter string is parsed using *strtok_r()*, each parameter is converted from string to binary by a call to:

```
strtoul(parameter, NULL, 16).
```

The format of the parameter string is:

```
device_addr:PCI_addr:ivec:ilevel:num_rds:num_tds:mem_base:mem_size:user_flags
```

TARGET-SPECIFIC PARAMETERS

device_addr

This is the base address at which the hardware device registers are located.

PCI_addr

This parameter defines the main memory address over the PCI bus. It is used to translate a physical memory address into a PCI-accessible address.

ivec

This is the interrupt vector number of the hardware interrupt generated by this Ethernet device. The driver uses *intConnect()* to attach an interrupt handler for this interrupt.

ilevel

This parameter defines the level of the hardware interrupt.

num_rds

The number of receive descriptors to use. This controls how much data the device can absorb under load. If this is specified as NONE (-1), the default of 32 is used.

num_tds

The number of transmit descriptors to use. This controls how much data the device can absorb under load. If this is specified as NONE (-1) then the default of 64 is used.

mem_base

This parameter specifies the base address of a DMA-able cache-free pre-allocated memory region for use as a memory pool for transmit/receive descriptors and

buffers. If there is no pre-allocated memory available for the driver, this parameter should be -1 (NONE). In which case, the driver allocates cache safe memory for its use using *cacheDmaAlloc()*.

mem_size

The memory size parameter specifies the size of the pre-allocated memory region. If memory base is specified as NONE (-1), the driver ignores this parameter.

user_flags

User flags control the run-time characteristics of the Ethernet chip. Most flags specify non default CSR0 and CSR6 bit values. Refer to **dec21x40End.h** for the bit values of the flags and to the device hardware reference manual for details about device capabilities, CSR6 and CSR0.

Device Type: Although the default device type is DEC 21040, specifying the **DEC_USR_21140** flag bit turns on DEC 21140 functionality.

Ethernet Address: The Ethernet address is retrieved from standard serial ROM on both DEC 21040, and DEC 21140 devices. If the retrieve from ROM fails, the driver calls the *sysDec21x40EnetAddrGet()* BSP routine. Specifying **DEC_USR_XEA** flag bit tells the driver should, by default, retrieve the Ethernet address using the *sysDec21x40EnetAddrGet()* BSP routine.

Priority RX processing: The driver programs the chip to process the transmit and receive queues at the same priority. By specifying **DEC_USR_BAR_RX**, the device is programmed to process receives at a higher priority.

TX poll rate: By default, the driver sets the Ethernet chip into a non-polling mode. In this mode, if the transmit engine is idle, it is kick-started every time a packet needs to be transmitted. Alternatively, the chip can be programmed to poll for the next available transmit descriptor if the transmit engine is in idle state. The poll rate is specified by one of **DEC_USR_TAP_xxx** flags.

Cache Alignment: The **DEC_USR_CAL_xxx** flags specify the address boundaries for data burst transfers.

DMA burst length: The **DEC_USR_PBL_xxx** flags specify the maximum number of long words in a DMA burst.

PCI multiple read: The **DEC_USR_RML** flag specifies that a device supports PCI memory-read-multiple.

Full Duplex Mode: When set, the **DEC_USR_FD** flag allows the device to work in full duplex mode, as long as the PHY used has this capability. Note that in this operation mode, the dec21x40 chip ignores the Collision and the Carrier Sense signals.

MII/Phy Checking: When set, and when a MII interface is being utilized the **DEC_USR_PHY_CHK** flag instructs the driver to wait until the PHY link status has changed to **up** before continuing. This time period could be as long as six seconds, but in general is on the order of two seconds. If clear, the check will not be performed. This option may be selected if the delay is unacceptable, but it is possible that a fast target may attempt to

send packets before the link is up. This will result in **no carrier** errors in packet transmission.

Transmit threshold value: The **DEC_USR_THR_XXX** flags enable the user to choose among different threshold values for the transmit FIFO. Transmission starts when the frame size within the transmit FIFO is larger than the threshold value. This should be selected taking into account the actual operating speed of the PHY. Again, see the device hardware reference manual for details.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires three external support functions and provides a hook function:

sysLanIntEnable()

```
void sysLanIntEnable (int level)
```

This routine provides a target-specific interface for enabling Ethernet device interrupts at a specified interrupt level.

sysLanIntDisable()

```
void sysLanIntDisable (void)
```

This routine provides a target-specific interface for disabling Ethernet device interrupts.

sysDec21x40EnetAddrGet()

```
STATUS sysDec21x40EnetAddrGet (int unit, char *enetAdrs)
```

This routine provides a target-specific interface for accessing a device Ethernet address.

_func_dec21x40MediaSelect

```
FUNCPTR _func_dec21x40MediaSelect
```

If *_func_dec21x40MediaSelect* is NULL, this driver provides a default media-select routine that reads and sets up physical media using the configuration information from a Version 3 DEC Serial ROM. Any other media configuration can be supported by initializing *_func_dec21x40MediaSelect*, typically in *sysHwInit()*, to a target-specific media select routine.

A media select routine is typically defined as:

```
STATUS decMediaSelect
(
    DEC21X40_DRV_CTRL *    pDrvCtrl,    /* driver control */
    UINT *                pCsr6Val     /* CSR6 return value */
)
{
    ...
}
```

The *pDrvCtrl* parameter is a pointer to the driver control structure that this routine can use to access the Ethernet device. The driver control structure member **mediaCount**, is initialized to 0xff at startup, while the other media control members (**mediaDefault**,

mediaCurrent, and **gprModeVal**) are initialized to zero. This routine can use these fields in any manner. However, all other driver control structure members should be considered read-only and should not be modified.

This routine should reset, initialize, and select an appropriate media. It should also write necessary the CSR6 bits (port select, PCS, SCR, and full duplex) to the memory location pointed to by *pCsr6Val*. The driver uses this value to program register CSR6. This routine should return OK or ERROR.

SEE ALSO

ifLib, *DECchip 21040 Ethernet LAN Controller for PCI*, *Digital Semiconductor 21140A PCI Fast Ethernet LAN Controller, Using the Digital Semiconductor 21140A with Boot ROM, Serial ROM, and External Register: An Application Note*

dhcpcBootLib

NAME

dhcpcBootLib – DHCP boot-time client library

ROUTINES

dhcpcBootInit() – set up the DHCP client parameters and data structures
dhcpcBootBind() – initialize the network with DHCP at boot time
dhcpcBootOptionSet() – add an option to the option request list

DESCRIPTION

This library contains the interface for the client side of the Dynamic Host Configuration Protocol (DHCP) used during system boot. DHCP is an extension of BOOTP, the bootstrap protocol. Like BOOTP, the protocol allows automatic system startup by providing an IP address, boot file name, and boot host's IP address over a network. Additionally, DHCP provides the complete set of configuration parameters defined in the Host Requirements RFCs and allows automatic reuse of network addresses by specifying a lease duration for a set of configuration parameters. This library is linked into the boot ROM image automatically if **INCLUDE_DHCP** is defined at the time that image is constructed.

HIGH-LEVEL INTERFACE

The VxWorks boot program uses this library to obtain configuration parameters with DHCP according to the client-server interaction detailed in RFC 1541 using the boot device specified in the boot parameters. The boot device must be capable of sending broadcast messages. Currently, only Ethernet devices and the shared-memory network drivers are supported. To use DHCP, first build a boot ROM image with **INCLUDE_DHCP** defined and set the appropriate flag in the boot parameters before initiating booting with the "@" command. The DHCP client will attempt to retrieve entries for the boot file name, host IP address, and target IP address, as well as a subnet mask and broadcast address for the boot device. Any entries retrieved will only be used if the corresponding fields in the boot parameters are blank.

dhcpcLib

- NOTE** After DHCP retrieves the boot parameters, the specified boot file is loaded and the system restarts. As a result, the boot-time DHCP client cannot renew the lease associated with the assigned IP address. To avoid potential IP address conflicts while loading the boot file, the `DHCPC_MIN_LEASE` value should be set to exceed the file transfer time. In addition, the boot file must also contain the DHCP client library so that the lease obtained before the restart can be renewed. Otherwise, the network initialization using the boot parameters will fail.
- INCLUDE FILES** `dhcpcBootLib.h`
- SEE ALSO** `dhcpcLib`, RFC 1541, RFC 1533

dhcpcLib

- NAME** `dhcpcLib` – Dynamic Host Configuration Protocol (DHCP) run-time client API
- ROUTINES**
- `dhcpcLibInit()` – DHCP client library initialization
 - `dhcpcInit()` – assign network interface and setup lease request
 - `dhcpcEventHookAdd()` – add a routine to handle configuration parameters
 - `dhcpcEventHookDelete()` – remove the configuration parameters handler
 - `dhcpcCacheHookAdd()` – add a routine to store and retrieve lease data
 - `dhcpcCacheHookDelete()` – delete a lease data storage routine
 - `dhcpcOptionSet()` – add an option to the option request list
 - `dhcpcBind()` – obtain a set of network configuration parameters with DHCP
 - `dhcpcVerify()` – renew an established lease
 - `dhcpcRelease()` – relinquish specified lease
 - `dhcpcShutdown()` – disable DHCP client library
 - `dhcpcOptionGet()` – retrieve an option provided to a client and store in a buffer
 - `dhcpcServerGet()` – retrieve the current DHCP server
 - `dhcpcTimerGet()` – retrieve current lease timers
 - `dhcpcParamsGet()` – retrieve current configuration parameters

- DESCRIPTION** This library implements the run-time access to the client side of the Dynamic Host Configuration Protocol (DHCP). DHCP is an extension of BOOTP. Like BOOTP, the protocol allows a host to initialize automatically by obtaining its IP address, boot file name, and boot host's IP address over a network. Additionally, DHCP provides a client with the complete set of parameters defined in the Host Requirements RFCs and allows automatic reuse of network addresses by specifying individual leases for each set of configuration parameters. The compatible message format allows DHCP participants to interact with BOOTP participants. The `dhcpcLibInit()` routine links this library into the VxWorks image. This happens automatically if `INCLUDE_DHCPC` is defined at the time the image is built.

CONFIGURATION INTERFACE

When used during run time, the DHCP client library establishes and maintains one or more DHCP leases. Each lease provides access to a set of configuration parameters. If requested, the parameters retrieved will be used to reconfigure the associated network interface, but may also be handled separately through an event hook. The *dhcpcEventHookAdd()* routine specifies a function which is invoked whenever the lease status changes. The *dhcpcEventHookDelete()* routine will disable that notification. The automatic reconfiguration must be limited to one lease for a particular network interface. Otherwise, multiple leases would attempt to reconfigure the same device, with unpredictable results.

HIGH-LEVEL INTERFACE

To access the DHCP client during run time, an application must first call the *dhcpcInit()* routine with a pointer to the network interface to be used for communication with a DHCP server. Each call to the initialization routine returns a unique identifier to be used in subsequent calls to the DHCP client routines. Next, the application must specify a client identifier for the lease using the *dhcpcOptionSet()* call. Typically, the link-level hardware address is used for this purpose. Additional calls to the option set routine may be used to request specific DHCP options. After all calls to that routine are completed, a call to *dhcpcBind()* will retrieve a set of configuration parameters according to the client-server interaction detailed in RFC 1541.

Each sequence of the three function calls described above, if successful, will retrieve a set of configuration parameters from a DHCP server. The *dhcpcServerGet()* routine retrieves the address of the server that provided a particular lease. The *dhcpcTimerGet()* routine will retrieve the current values for both lease timers.

Alternatively, the *dhcpcParamsGet()* and *dhcpcOptionGet()* routines will access any options provided by a DHCP server. In addition to the lease identifier obtained from the initialization routine, the *dhcpcParamsGet()* routine accepts a parameter descriptor structure that selects any combination of the options described in RFC 1533 for retrieval. Similarly, *dhcpcOptionGet()* retrieves the values associated with a single option.

LOW-LEVEL INTERFACE

This library also contains several routines which explicitly generate DHCP messages. *dhcpcVerify()* causes the client to renew a particular lease, regardless of the time remaining. *dhcpcRelease()* relinquishes the specified lease. The associated parameters are no longer valid. If those parameters were used by the underlying network device, the routine also shuts off all network processing for that interface. Finally, *dhcpcShutdown()* releases all active leases and disables all the DHCP client library routines.

OPTIONAL INTERFACE

The *dhcpcCacheHookAdd()* routine registers a function that the client will use to store and retrieve lease data. The client can then re-use this information if it is rebooted. The *dhcpcCacheHookDelete()* routine prevents the re-use of lease data. Initially, a function to access permanent storage is not provided.

INCLUDE FILES **dhcpcLib.h**

SEE ALSO RFC 1541, RFC 1533

dhcpcShow

NAME **dhcpcShow** – DHCP run-time client information display routines

ROUTINES **dhcpcShowInit()** – initialize the DHCP show facility
dhcpcServerShow() – display current DHCP server
dhcpcTimersShow() – display current lease timers
dhcpcParamsShow() – display current lease parameters

DESCRIPTION This library provides routines that display various information related to the DHCP run-time client library such as the lease timers and responding server. The **dhcpcShowInit()** routine links the show facility into the VxWorks image. This happens automatically if **INCLUDE_NET_SHOW** and **INCLUDE_DHCP** are defined at the time the image is built.

INCLUDE FILES **dhcpcLib.h**

SEE ALSO **dhcpcLib**, *Network Programmer's Guide: Network Configuration Protocols*

dhcprLib

NAME **dhcprLib** – DHCP relay agent library

ROUTINES No Callable Routines

DESCRIPTION This library implements a relay agent for the Dynamic Host Configuration Protocol (DHCP). DHCP is an extension of BOOTP. Like BOOTP, it allows a target to configure itself dynamically by using the network to get its IP address, a boot file name, and the DHCP server's address. The relay agent forwards DHCP messages between clients and servers resident on different subnets. The standard DHCP server, if present on a subnet, can also forward messages across subnet boundaries. The relay agent is needed only if there is no DHCP server running on the subnet. The **dhcprLibInit()** routine links this library into the VxWorks system. This happens automatically if **INCLUDE_DHCP** is defined at the time the system is built, as long as **INCLUDE_DHCP** is *not* also defined.

HIGH-LEVEL INTERFACE

The *dhcprInit()* routine initializes the relay agent automatically. The relay agent forwards incoming DHCP messages to the IP addresses specified at build time in the *dhcpTargetTbl[]* array in *usrNetwork.c*.

INCLUDE FILES *dhcprLib.h*

SEE ALSO RFC 1541, RFC 1533

dhcpsLib

NAME *dhcpsLib* – Dynamic Host Configuration Protocol (DHCP) server library

ROUTINES *dhcpsInit()* – set up the DHCP server parameters and data structures
dhcpsLeaseEntryAdd() – add another entry to the address pool
dhcpsLeaseHookAdd() – assign a permanent lease storage hook for the server
dhcpsAddressHookAdd() – assign a permanent address storage hook for the server

DESCRIPTION This library implements the server side of the Dynamic Host Configuration Protocol (DHCP). DHCP is an extension of BOOTP. Like BOOTP, it allows a target to configure itself dynamically by using the network to get its IP address, a boot file name, and the DHCP server's address. Additionally, DHCP provides for automatic reuse of network addresses by specifying individual leases as well as many additional options. The compatible message format allows DHCP participants to interoperate with BOOTP participants. The *dhcpsInit()* routine links this library into the VxWorks image. This happens automatically if **INCLUDE_DHCP**S is defined when the image is built.

PRIMARY INTERFACE

The *dhcpsInit()* routine initializes the server. It reads the hard-coded server configuration data that is stored in three separate tables in *usrNetwork.c*. The first table contains entries as follows:

```
DHCPS_LEASE_DESC dhcpsLeaseTbl [] =
{
  {"sample1", "90.11.42.24", "90.11.42.24", "clid=\\"1:0x08003D21FE90\\"},
  {"sample2", "90.11.42.25", "90.11.42.28", "maxl=90:dfll=60"},
  {"sample3", "90.11.42.29", "90.11.42.34",
   "maxl=0xffffffff:file=/vxWorks"},
  {"sample4", "90.11.42.24", "90.11.42.24", "albp=true:file=/vxWorks"}
};
```

Each entry contains a name of up to eight characters, the starting and ending IP addresses of a range, and the parameters associated with the lease. The four samples shown demonstrate the four types of leases.

Manual leases contain a specific client ID, and are issued only to that client, with an infinite duration. The example shown specifies a MAC address, which is the identifier type used by the VxWorks DHCP client.

Dynamic leases specify a finite maximum length, and can be issued to any requesting client. These leases allow later re-use of the assigned IP address. If not explicitly specified in the parameters field, these leases use the values of **DHCPS_MAX_LEASE** and **DHCPS_DFLT_LEASE** to determine the lease length.

Automatic leases are implied by the infinite maximum length. Their IP addresses are assigned permanently to any requesting client.

The last sample demonstrates a lease that is also available to BOOTP clients. The infinite maximum length is implied, and any timing-related parameters are ignored.

The DHCP server supplies leases to DHCP clients according to the lease type in the order shown above. Manual leases have the highest priority and leases available to BOOTP clients the lowest.

Entries in the parameters field may be one of these types:

bool

Takes values of "true" or "false", for example, ipfd=true. Unrecognized values default to false.

str

Takes a character string as a value, for example, hstn="clapton". If the string includes a delimiter character, such as a colon, it should be enclosed in quotation marks.

octet

Takes an 8-bit integer in decimal, octal, or hexadecimal, for example, 8, 070, 0xff.

short

Takes a 16-bit integer.

long

Takes a 32-bit integer.

ip

Takes a string that is interpreted as a 32-bit IP address. One of the following formats is expected: **a.b.c.d**, **a.b.c** or **a.b**. In the second format, c is interpreted as a 16-bit value. In the third format, b is interpreted as a 24-bit value, for example siad=90.11.42.1.

iplist

Takes a list of IP addresses, separated by white space, for example, rout=133.4.31.1 133.4.31.2 133.4.31.3.

ippairs

Takes a list of IP address pairs. Each IP address is separated by white space and grouped in pairs, for example, `strt=133.4.27.0 133.4.31.1 133.4.36.0 133.4.31.1`.

mtpt

Takes a list of 16 bit integers, separated by white space, for example, `mtpt=1 2 3 4 6 8`.

clid

Takes a client identifier as a value. Client identifiers are represented by the quoted string "*type:data*", where *type* is an integer from 0 to 255, as defined by the IANA, and *data* is a sequence of 8-bit values in hexadecimal. The client ID is usually a MAC address, for example, `clid="1:0x08004600e5d5"`.

The following table lists the option specifiers and descriptions for every possible entry in the parameter list. When available, the option code from RFC 1533 is included.

| Name | Code | Type | Description |
|--------------------|------|--------|---|
| <code>sname</code> | - | str | Optional server name. |
| <code>file</code> | - | str | Name of file containing the boot image. |
| <code>siad</code> | - | ip | Address of server that offers the boot image. |
| <code>albp</code> | - | bool | If true, this entry is also available to BOOTP clients. For entries using static allocation, this value becomes true by default and <i>maxl</i> becomes infinity. |
| <code>maxl</code> | - | long | Maximum lease duration in seconds. |
| <code>dfl</code> | - | long | Default lease duration in seconds. If a client does not request a specific lease duration, the server uses this value. |
| <code>clid</code> | - | clid | This specifies a client identifier for manual leases. The VxWorks client uses a MAC address as the client identifier. |
| <code>pmid</code> | - | clid | This specifies a client identifier for client-specific parameters to be included in a lease. It should be present in separate entries without IP addresses. |
| <code>clas</code> | - | str | This specifies a class identifier for class-specific parameters to be included in a lease. It should be present in separate entries without IP addresses. |
| <code>snmk</code> | 1 | ip | Subnet mask of the IP address to be allocated. The default is a natural mask corresponding to the IP address. The server will not issue IP addresses to clients on different subnets. |
| <code>tmof</code> | 2 | long | Time offset from UTC in seconds. |
| <code>rout</code> | 3 | iplist | A list of routers on the same subnet as the client. |
| <code>tmsv</code> | 4 | iplist | A list of time servers (RFC 868). |
| <code>nmsv</code> | 5 | iplist | A list of name servers (IEN 116). |
| <code>dnsv</code> | 6 | iplist | A list of DNS servers (RFC 1035). |
| <code>lgsv</code> | 7 | iplist | A list of MIT-LCS UDP log servers. |
| <code>cksv</code> | 8 | iplist | A list of Cookie servers (RFC 865). |
| <code>lpsv</code> | 9 | iplist | A list of LPR servers (RFC 1179). |

| Name | Code | Type | Description |
|-------------|-------------|-------------|---|
| imsv | 10 | iplist | A list of Imagen Impress servers. |
| rlsv | 11 | iplist | A list of Resource Location servers (RFC 887). |
| hstn | 12 | str | Hostname of the client. |
| btsz | 13 | short | Size of boot image. |
| mdmp | 14 | str | Path name to which client dumps core. |
| dnsd | 15 | str | Domain name for DNS. |
| swsv | 16 | ip | IP address of swap server. |
| rpth | 17 | str | Path name of root disk of the client. |
| epth | 18 | str | Extensions Path (See RFC 1533). |
| ipfd | 19 | bool | If true, the client performs IP forwarding. |
| nlsr | 20 | bool | If true, the client can perform non-local source routing. |
| plcy | 21 | ippairs | Policy filter for non-local source routing. A list of pairs of (Destination IP, Subnet mask). |
| mdgs | 22 | short | Maximum size of IP datagram that the client should be able to reassemble. |
| ditl | 23 | octet | Default IP TTL. |
| mtat | 24 | long | Aging timeout (in seconds) to be used with Path MTU discovery (RFC 1191). |
| mtpt | 25 | mtpt | A table of MTU sizes to be used with Path MTU Discovery. |
| ifmt | 26 | short | MTU to be used on an interface. |
| asnl | 27 | bool | If true, the client assumes that all subnets to which the client is connected use the same MTU. |
| brda | 28 | ip | Broadcast address in use on the client's subnet. The default is calculated from the subnet mask and the IP address. |
| mskd | 29 | bool | If true, the client should perform subnet mask discovery using ICMP. |
| msks | 30 | bool | If true, the client should respond to subnet mask requests using ICMP. |
| rtrd | 31 | bool | If true, the client should solicit routers using Router Discovery defined in RFC 1256. |
| rtsl | 32 | ip | Destination IP address to which the client sends router solicitation requests. |
| strt | 33 | ippairs | A table of static routes for the client, which are pairs of (Destination, Router). It is illegal to specify default route as a destination. |
| trlr | 34 | bool | If true, the client should negotiate the use of trailers with ARP (RFC 893). |
| arpt | 35 | long | Timeout in seconds for ARP cache. |
| encp | 36 | bool | If false, the client uses RFC 894 encapsulation. If true, it uses RFC 1042 (IEEE 802.3) encapsulation. |
| dttl | 37 | octet | Default TTL of TCP. |

| Name | Code | Type | Description |
|------|------|--------|---|
| kain | 38 | long | Interval of the client's TCP keepalive in seconds. |
| kagb | 39 | bool | If true, the client should send TCP keepalive messages with a octet of garbage for compatibility. |
| nisd | 40 | str | Domain name for NIS. |
| nisv | 41 | iplist | A list of NIS servers. |
| ntsv | 42 | iplist | A list of NTP servers. |
| nnsv | 44 | iplist | A list of NetBIOS name server. (RFC 1001, 1002) |
| ndsv | 45 | iplist | A list of NetBIOS datagram distribution servers (RFC 1001, 1002). |
| nbnt | 46 | octet | NetBIOS node type (RFC 1001, 1002). |
| nbsc | 47 | str | NetBIOS scope (RFC 1001, 1002). |
| xfsv | 48 | iplist | A list of font servers of X Window system. |
| xdmn | 49 | iplist | A list of display managers of X Window system. |
| dht1 | 58 | short | This value specifies when the client should start RENEWING. The default of 500 means the client starts RENEWING after 50% of the lease duration passes. |
| dht1 | 59 | short | This value specifies when the client should start REBINDING. The default of 875 means the client starts REBINDING after 87.5% of the lease duration passes. |

Finally, to function correctly, the DHCP server requires access to some form of permanent storage. The `DHCPS_LEASE_HOOK` constant specifies the name of a storage routine with the following interface:

```
STATUS dhcpsStorageHook (int op, char *buffer, int datalen);
```

The storage routine is installed by a call to the `dhcpsLeaseHookAdd()` routine. The manual pages for `dhcpsLeaseHookAdd()` describe the parameters and required operation of the storage routine.

SECONDARY INTERFACE

In addition to the hard-coded entries, address entries may be added after the server has started by calling the following routine:

```
STATUS dhcpsLeaseEntryAdd (char *name, char *start, char *end, char *config);
```

The parameters specify an entry name, starting and ending values for a block of IP addresses, and additional configuration information in the same format as shown above for the hard-coded entries. Each parameter must be formatted as a NULL-terminated string.

The `DHCPS_ADDRESS_HOOK` constant specifies the name of a storage routine, used to preserve address entries added after startup, which has the following prototype:

```
STATUS dhcpsAddressStorageHook (int op, char *name, char *start, char *end,  
                                char *params);
```

The storage routine is installed with the *dhcpsAddressHookAdd()* routine, and is fully described in the manual pages for that function.

OPTIONAL INTERFACE

The DHCP server can also receive messages forwarded from different subnets by a relay agent. To provide addresses to clients on different subnets, the appropriate relay agents must be listed in the provided table in **usrNetwork.c**. A sample configuration is:

```
DHCPS_RELAY_DESC dhcpsRelayTbl [] =
{
    {"90.11.46.75", "90.11.46.0"}
};
```

Each entry in the table specifies the address of a relay agent that will transmit the request and the corresponding subnet number. To issue leases successfully, the address pool must also contain IP addresses for the monitored subnets.

The following table allows a DHCP server to act as a relay agent in addition to its default function of processing messages. It consists of a list of IP addresses.

```
DHCP_TARGET_DESC dhcpTargetTbl [] =
{
    {"90.11.43.2"},
    {"90.11.44.1"}
};
```

Each IP address in this list receives a copy of any client messages generated on the subnets monitored by the server.

INCLUDE FILES **dhcpsLib.h**

SEE ALSO RFC 1541, RFC 1533

dirLib

NAME **dirLib** – directory handling library (POSIX)

ROUTINES

- opendir()* – open a directory for searching (POSIX)
- readdir()* – read one entry from a directory (POSIX)
- rewinddir()* – reset position to the start of a directory (POSIX)
- closedir()* – close a directory (POSIX)
- fstat()* – get file status information (POSIX)
- stat()* – get file status information using a pathname (POSIX)
- fstatfs()* – get file status information (POSIX)

statfs() – get file status information using a pathname (POSIX)

utime() – update time on a file

DESCRIPTION

This library provides POSIX-defined routines for opening, reading, and closing directories on a file system. It also provides routines to obtain more detailed information on a file or directory.

SEARCHING DIRECTORIES

Basic directory operations, including *opendir()*, *readdir()*, *rewinddir()*, and *closedir()*, determine the names of files and subdirectories in a directory.

A directory is opened for reading using *opendir()*, specifying the name of the directory to be opened. The *opendir()* call returns a pointer to a directory descriptor, which identifies a directory stream. The stream is initially positioned at the first entry in the directory.

Once a directory stream is opened, *readdir()* is used to obtain individual entries from it. Each call to *readdir()* returns one directory entry, in sequence from the start of the directory. The *readdir()* routine returns a pointer to a **dirent** structure, which contains the name of the file (or subdirectory) in the **d_name** field.

The *rewinddir()* routine resets the directory stream to the start of the directory. After *rewinddir()* has been called, the next *readdir()* will cause the current directory state to be read in, just as if a new *opendir()* had occurred. The first entry in the directory will be returned by the first *readdir()*.

The directory stream is closed by calling *closedir()*.

GETTING FILE INFORMATION

The directory stream operations described above provide a mechanism to determine the names of the entries in a directory, but they do not provide any other information about those entries. More detailed information is provided by *stat()* and *fstat()*.

The *stat()* and *fstat()* routines are essentially the same, except for how the file is specified. The *stat()* routine takes the name of the file as an input parameter, while *fstat()* takes a file descriptor number as returned by *open()* or *creat()*. Both routines place the information from a directory entry in a **stat** structure whose address is passed as an input parameter. This structure is defined in the include file **stat.h**. The fields in the structure include the file size, modification date/time, whether it is a directory or regular file, and various other values.

The **st_mode** field contains the file type; several macro functions are provided to test the type easily. These macros operate on the **st_mode** field and evaluate to TRUE or FALSE depending on whether the file is a specific type. The macro names are:

S_ISREG

test if the file is a regular file

S_ISDIR

test if the file is a directory

dosFsLib**S_ISCHR**

test if the file is a character special file

S_ISBLK

test if the file is a block special file

S_ISFIFO

test if the file is a FIFO special file

Only the regular file and directory types are used for VxWorks local file systems. However, the other file types may appear when getting file status from a remote file system (using NFS).

As an example, the **S_ISDIR** macro tests whether a particular entry describes a directory. It is used as follows:

```
char      *filename;
struct stat fileStat;
stat (filename, &fileStat);
if (S_ISDIR (fileStat.st_mode))
    printf ("%s is a directory.\n", filename);
else
    printf ("%s is not a directory.\n", filename);
```

See the *ls()* routine in **usrLib** for an illustration of how to combine the directory stream operations with the *stat()* routine.

INCLUDE FILES **dirent.h, stat.h**

dosFsLib

NAME **dosFsLib** – MS-DOS media-compatible file system library

ROUTINES *dosFsConfigGet()* – obtain dosFs volume configuration values
dosFsConfigInit() – initialize dosFs volume configuration structure
dosFsConfigShow() – display dosFs volume configuration data
dosFsDateSet() – set the dosFs file system date
dosFsDateTimeInstall() – install a user-supplied date/time function
dosFsDevInit() – associate a block device with dosFs file system functions
dosFsDevInitOptionsSet() – specify volume options for *dosFsDevInit()*
dosFsInit() – prepare to use the dosFs library
dosFsMkfs() – initialize a device and create a dosFs file system
dosFsMkfsOptionsSet() – specify volume options for *dosFsMkfs()*
dosFsModeChange() – modify the mode of a dosFs volume
dosFsReadyChange() – notify dosFs of a change in ready status

dosFsTimeSet() – set the dosFs file system time
dosFsVolOptionsGet() – get current dosFs volume options
dosFsVolOptionsSet() – set dosFs volume options
dosFsVolUnmount() – unmount a dosFs volume

DESCRIPTION This library provides services for file-oriented device drivers to use the MS-DOS[®] file standard. This module takes care of all necessary buffering, directory maintenance, and file system details.

USING THIS LIBRARY

The various routines provided by the VxWorks DOS file system (dosFs) may be separated into three broad groups: general initialization, device initialization, and file system operation.

The *dosFsInit()* routine is the principal initialization function; it need only be called once, regardless of how many dosFs devices are to be used. In addition, *dosFsDateTimeInstall()* (if used) will typically be called only once, prior to performing any actual file operations, to install a user-supplied routine which provides the current date and time.

Other dosFs functions are used for device initialization. For each dosFs device, either *dosFsDevInit()* or *dosFsMkfs()* must be called to install the device and define its configuration. The *dosFsConfigInit()* routine is provided to easily initialize the data structure used during device initialization; however, its use is optional.

Several routines are provided to inform the file system of changes in the system environment. The *dosFsDateSet()* and *dosFsTimeSet()* routines are used to set the current date and time; these are normally used only if no user routine has been installed via *dosFsDateTimeInstall()*. The *dosFsModeChange()* call may be used to modify the readability or writability of a particular device. The *dosFsReadyChange()* routine is used to inform the file system that a disk may have been swapped, and that the next disk operation should first remount the disk. Finally, *dosFsVolUnmount()* informs the file system that a particular device should be synchronized and unmounted, generally in preparation for a disk change.

More detailed information on all of these routines is discussed in the following sections.

INITIALIZING DOSFSLIB

Before any other routines in **dosFsLib** can be used, the routine *dosFsInit()* must be called to initialize this library. This call specifies the maximum number of dosFs files that can be open simultaneously. Attempts to open more dosFs files than the specified maximum will result in errors from *open()* and *creat()*.

This initialization is enabled when the configuration macro **INCLUDE_DOSFS** is defined; *dosFsInit()* is then called from the root task, *usrRoot()*, in *usrConfig.c*.

DEFINING A DOSFS DEVICE

To use this library for a particular device, the device descriptor structure used by the

dosFsLib

device driver must contain, as the very first item, a block device description structure (**BLK_DEV**). This must be initialized before calling *dosFsDevInit()*. In the **BLK_DEV** structure, the driver includes the addresses of five routines which it must supply: one that reads one or more sectors, one that writes one or more sectors, one that performs I/O control on the device (using *ioctl()*), one that checks the status of the device, and one that resets the device. These routines are described below. The **BLK_DEV** structure also contains fields which describe the physical configuration of the device. For more information about defining block devices, see the *VxWorks Programmer's Guide: I/O System*.

The *dosFsDevInit()* routine associates a device with the **dosFsLib** functions. It expects three parameters:

- (1) A pointer to a name string, to be used to identify the device. This will be part of the pathname for I/O operations which operate on the device. This name will appear in the I/O system device table, which may be displayed using the *iosDevShow()* routine.
- (2) A pointer to the **BLK_DEV** structure which describes the device and contains the addresses of the five required functions. The fields in this structure must have been initialized before the call to *dosFsDevInit()*.
- (3) A pointer to a volume configuration structure (**DOS_VOL_CONFIG**). This structure contains configuration data for the volume which are specific to the dosFs file system. (See "Changes in Volume Configuration", below, for more information.) The fields in this structure must have been initialized before the call to *dosFsDevInit()*. The **DOS_VOL_CONFIG** structure may be initialized by using the *dosFsConfigInit()* routine.

As an example:

```
dosFsDevInit
(
  char          *volName,      /* name to be used for volume */
  BLK_DEV       *pBlkDev,     /* pointer to device descriptor */
  DOS_VOL_CONFIG *pVolConfig  /* pointer to vol config data */
)
```

Once *dosFsDevInit()* has been called, when **dosFsLib** receives a request from the I/O system, it calls the device driver routines (whose addresses were passed in the **BLK_DEV** structure) to access the device.

The *dosFsMkfs()* routine is an alternative to using *dosFsDevInit()*. The *dosFsMkfs()* routine always initializes a new dosFs file system on the disk; thus, it is unsuitable for disks containing data that should be preserved. Default configuration parameters are supplied by *dosFsMkfs()*, since no **DOS_VOL_CONFIG** structure is used.

See "Network File System (NFS) Support", below, for additional NFS-related parameters you can set before calling *dosFsDevInit()*.

MULTIPLE LOGICAL DEVICES

The sector number passed to the driver's sector read and write routines is an absolute number, starting from sector 0 at the beginning of the device. If desired, the driver may add an offset from the beginning of the physical device before the start of the logical device. This can be done by keeping an offset parameter in the driver device structure, and adding the offset to the sector number passed by the file system's read and write routines.

ACCESSING THE RAW DISK

As a special case in *open()* and *creat()* calls, the dosFs file system recognizes a null filename as indicating access to the entire "raw" disk rather than to an individual file on the disk. (To open a device in raw mode, specify only the device name -- no filename -- during the *open()* or *creat()* call.)

Raw mode is the only means of accessing a disk that has no file system. For example, to initialize a new file system on the disk, first the raw disk is opened and the returned file descriptor is used for an *ioctl()* call with **FIODISKINIT**. Opening the disk in raw mode is also a common operation when doing other *ioctl()* functions which do not involve a particular file (e.g., **FIONFREE**, **FIOLABELGET**).

To read the root directory of a disk on which no file names are known, specify the device name when calling *opendir()*. Subsequent *readdir()* calls will return the names of files and subdirectories in the root directory.

Data written to the disk in raw mode uses the same area on the disk as normal dosFs files and subdirectories. Raw I/O does not use the disk sectors used for the boot sector, root directory, or File Allocation Table (FAT). For more information about raw disk I/O using the entire disk, see the manual entry for **rawFsLib**.

DEVICE AND PATH NAMES

On true MS-DOS machines, disk device names are typically of the form "A:", that is, a single letter designator followed by a colon. Such names may be used with the VxWorks dosFs file system. However, it is possible (and desirable) to use longer, more mnemonic device names, such as "DOS1:", or "/floppy0/". The name is specified during the *dosFsDevInit()* or *dosFsMkfs()* call.

The pathnames used to specify dosFs files and directories may use either forward slashes ("/") or backslashes ("\") effect on the directory data written to the disk. (Note, however, that forward slashes are not allowed within VxWorks dosFs filenames, although they are normally legal for pure MS-DOS implementations.)

When using the VxWorks shell to make calls specifying dosFs pathnames, you must allow for the C-style interpretation which is performed. In cases where the file name is enclosed in quote marks, any backslashes must be "escaped" by a second, preceding backslash. For example:

```
-> copy ("DOS1:\\subdir\\file1", "file2")
```

However, shell commands which use pathnames without enclosing quotes do not require the second backslash. For example:

```
-> copy < DOS1:\subdir\file1
```

Forward slashes do not present these inconsistencies, and may therefore be preferable for use within the shell.

The leading slash of a dosFs pathname following the device name is optional. For example, both "DOS1:newfile.new" and "DOS1:/newfile.new" refer to the same file.

USING EXTENDED FILE NAMES

The MS-DOS standard only allows for file names which fit the restrictions of eight upper-case characters optionally followed by a three-character extension. This may not be convenient if you are transferring files to or from a remote system, or if your application requires particular file naming conventions.

To provide additional flexibility, the dosFs file system provides an option to use longer, less restricted file names. When this option is enabled, file names may consist of any sequence of up to 40 ASCII characters. No case conversion is performed and no characters have any special significance.

NOTE

Because special directory entries are used on the disk, disks which use the extended names are *not* compatible with true MS-DOS systems and cannot be read on MS-DOS machines. Disks which use the extended name option must be initialized by the VxWorks dosFs file system (using **FIODISKINIT**); disks which have been initialized (software-formatted) on MS-DOS systems cannot be used.

To enable the extended file names, set the **DOS_OPT_LONGNAMES** bit in the **dosvc_options** field in the **DOS_VOL_CONFIG** structure when calling *dosFsDevInit()*. (The *dosFsMkfs()* routine may also be used to enable extended file names; however, the **DOS_OPT_LONGNAMES** option must already have been specified in a previous call to *dosFsMkfsOptionsSet()*.)

NETWORK FILE SYSTEM (NFS) SUPPORT

To enable the export of a file system, the **DOS_OPT_EXPORT** option must be set when initializing the device via *dosFsDevInit()* or *dosFsMkfs()*. This option may also be made the default for use with disks when no explicit configuration is given. See the manual entry for *dosFsDevInitOptionsSet()*.

If the remote client that will be mounting the dosFs volume is a PC-based client, you may also need to specify the **DOS_OPT_LOWERCASE** option. This option causes filenames to be mapped to lowercase (when not using the **DOS_OPT_LONGNAMES** option). This lowercase mapping is expected by many PC-based NFS implementations.

When the **DOS_OPT_EXPORT** option is enabled, the VxWorks NFS file system uses the reserved fields of a dosFs directory entry to store information needed to uniquely identify a dosFs file.

Every time a file is created in a directory, the directory timestamp is incremented. This is necessary to avoid cache inconsistencies in clients, because some UNIX clients use the directory timestamp to determine if their local cache needs to be updated.

You can also specify integers for a user ID, group ID, and file access permissions byte when you initialize a dosFs file system for NFS export. The values you specify will apply to all files in the file system.

Set **dosFsUserId** to specify the numeric user ID. The default is 65534.

Set **dosFsGroupId** to specify the numeric group ID. The default is 65534.

Set **dosFsFileMode** to specify the numeric file access mode. The default is 777.

READING DIRECTORY ENTRIES

Directories on VxWorks dosFs volumes may be searched using the *opendir()*, *readdir()*, *rewinddir()*, and *closedir()* routines. These calls allow the names of files and subdirectories to be determined.

To obtain more detailed information about a specific file, use the *fstat()* or *stat()* routine. Along with standard file information, the structure used by these routines also returns the file attribute byte from a dosFs directory entry.

For more information, see the manual entry for **dirLib**.

FILE DATE AND TIME

Directory entries on dosFs volumes contain a time and date for each file or subdirectory. This time is set when the file is created, and it is updated when a file is closed, if it has been modified. Directory time and date fields are set only when the directory is created, not when it is modified.

The dosFs file system library maintains the date and time in an internal structure. While there is currently no mechanism for automatically advancing the date or time, two different methods for setting the date and time are provided.

The first method involves using two routines, *dosFsDateSet()* and *dosFsTimeSet()*, which are provided to set the current date and time.

Examples of setting the date and time would be:

```
dosFsDateSet (1990, 12, 25); /* set date to Dec-25-1990 */
dosFsTimeSet (14, 30, 22); /* set time to 14:30:22 */
```

The second method requires a user-provided hook routine. If a time and date hook routine is installed using *dosFsDateTimeInstall()*, the routine will be called whenever **dosFsLib** requires the current date. This facility is provided to take advantage of hardware time-of-day clocks which may be read to obtain the current time.

The date/time hook routine should be defined as follows:

```
void dateTimeHook
(
```

dosFsLib

```

        DOS_DATE_TIME    *pDateTime    /* ptr to dosFs date/time struct */
    )

```

On entry to the hook routine, the `DOS_DATE_TIME` structure will contain the last time and date which was set in `dosFsLib`. The structure should then be filled by the hook routine with the correct values for the current time and date. Unchanged fields in the structure will retain their previous values.

The MS-DOS specification only provides for 2-second granularity for file time stamps. If the number of seconds in the time specified during `dosFsTimeSet()` or the date/time hook routine is odd, it will be rounded down to the next even number.

The date and time used by `dosFsLib` is initially Jan-01-1980, 00:00:00.

FILE ATTRIBUTES Directory entries on dosFs volumes contain an attribute byte consisting of bit-flags which specify various characteristics of the entry. The attributes which are identified are: read-only file, hidden file, system file, volume label, directory, and archive. The VxWorks symbols for these attribute bit-flags are:

```

    DOS_ATTR_RDONLY
    DOS_ATTR_HIDDEN
    DOS_ATTR_SYSTEM
    DOS_ATTR_VOL_LABEL
    DOS_ATTR_DIRECTORY
    DOS_ATTR_ARCHIVE

```

All the flags in the attribute byte, except the directory and volume label flags, may be set or cleared using the `ioctl()` `FIOATTRIBSET` function. This function is called after opening the specific file whose attributes are to be changed. The attribute byte value specified in the `FIOATTRIBSET` call is copied directly. To preserve existing flag settings, the current attributes should first be determined via `fstat()`, and the appropriate flag(s) changed using bitwise AND or OR operations. For example, to make a file read-only, while leaving other attributes intact:

```

    struct stat fileStat;
    fd = open ("file", O_RDONLY, 0);    /* open file          */
    fstat (fd, &fileStat);             /* get file status    */
    ioctl (fd, FIOATTRIBSET, (fileStat.st_attr | DOS_ATTR_RDONLY));
                                        /* set read-only flag */
    close (fd);                        /* close file         */

```

CONTIGUOUS FILE SUPPORT

The VxWorks dosFs file system provides efficient handling of contiguous files, meaning files which are made up of a consecutive series of disk sectors. This support includes both the ability to allocate contiguous space to a file (or directory) and optimized access to such a file when it is used.

To allocate a contiguous area to a file, the file is first created in the normal fashion, using `open()` or `creat()`. The file descriptor returned during the creation of the file is then used

to make an *ioctl()* call, specifying the **FIOCONTIG** function. The other parameter to the **FIOCONTIG** function is the size of the requested contiguous area in bytes. It is also possible to request that the largest contiguous free area on the disk be obtained. In this case, the special value **CONTIG_MAX** (-1) is used instead of an actual size.

The FAT is searched for a suitable section of the disk, and if found, it is assigned to the file. (If there is no contiguous area on the volume large enough to satisfy the request, an **S_dosFsLib_NO_CONTIG_SPACE** error is returned.) The file may then be closed or used for further I/O operations. For example, the following will create a file and allocate 0x10000 contiguous bytes:

```
fd = creat ("file", O_RDWR, 0);           /* open file          */
status = ioctl (fd, FIOCONTIG, 0x10000); /* get contiguous area */
if (status != OK)
    ...                                   /* do error handling  */
close (fd);                             /* close file         */
```

In contrast, the following example will create a file and allocate the largest contiguous area on the disk to it:

```
fd = creat ("file", O_RDWR, 0);           /* open file          */
status = ioctl (fd, FIOCONTIG, CONTIG_MAX); /* get contiguous area */
if (status != OK)
    ...                                   /* do error handling  */
close (fd);                             /* close file         */
```

It is important that the file descriptor used for the *ioctl()* call be the only descriptor open to the file. Furthermore, since a file may be assigned a different area of the disk than was originally allocated, the **FIOCONTIG** operation should take place before any data is written to the file.

To determine the actual amount of contiguous space obtained when **CONTIG_MAX** is specified as the size, use *fstat()* to examine the file size. For more information, see **dirLib**.

Space which has been allocated to a file may later be freed by using *ioctl()* with the **FIOTRUNC** function.

Directories may also be allocated a contiguous disk area. A file descriptor to the directory is used to call **FIOCONTIG**, just as for a regular file. A directory should be empty (except for the "." and ".." entries) before it has contiguous space allocated to it. The root directory allocation may not be changed. Space allocated to a directory is not reclaimed until the directory is deleted; directories may not be truncated using the **FIOTRUNC** function.

When any file is opened, it is checked for contiguity. If a file is recognized as contiguous, more efficient techniques for locating specific sections of the file are used, rather than following cluster chains in the FAT as must be done for fragmented files. This enhanced handling of contiguous files takes place regardless of whether the space was actually allocated using **FIOCONTIG**.

CHANGING, UNMOUNTING, AND SYNCHRONIZING DISKS

Copies of directory entries and the FAT for each volume are kept in memory. This greatly speeds up access to files, but it requires that **dosFsLib** be notified when disks are changed (i.e., floppies are swapped). Two different notification mechanisms are provided.

Unmounting Volumes

The first, and preferred, method of announcing a disk change is for *dosFsVolUnmount()* to be called prior to removal of the disk. This call flushes all modified data structures to disk, if possible (see the description of disk synchronization below), and also marks any open file descriptors as obsolete. During the next I/O operation, the disk is remounted. The *ioctl()* call may also be used to initiate *dosFsVolUnmount()*, by specifying the **FIOUNMOUNT** function code. (Any open file descriptor to the device may be used in the *ioctl()* call.)

There may be open files or directories on a dosFs volume when it is unmounted. If this is the case, those file descriptors will be marked as obsolete. Any attempts to use them for further I/O operations will return an **S_dosFsLib_FD_OBSOLETE** error. To free such file descriptors, use the *close()* call, as usual. This will successfully free the descriptor, but will still return **S_dosFsLib_FD_OBSOLETE**. File descriptors acquired when opening the entire volume (raw mode) will not be marked as obsolete during *dosFsVolUnmount()* and may still be used.

Interrupt handlers must not call *dosFsVolUnmount()* directly, because it is possible for the *dosFsVolUnmount()* call to block while the device becomes available. The interrupt handler may instead give a semaphore which readies a task to unmount the volume. (Note that *dosFsReadyChange()* may be called directly from interrupt handlers.)

When *dosFsVolUnmount()* is called, it attempts to write buffered data out to the disk. It is therefore inappropriate for situations where the disk change notification does not occur until a new disk has been inserted. (The old buffered data would be written to the new disk.) In these circumstances, *dosFsReadyChange()* should be used.

If *dosFsVolUnmount()* is called after the disk is physically removed (i.e., there is no disk in the drive), the data-flushing portion of its operation will fail. However, the file descriptors will still be marked as obsolete, and the disk will be marked as requiring remounting. An error will not be returned by *dosFsVolUnmount()* in this situation. To avoid lost data in such a situation, the disk should be explicitly synchronized before it is removed.

Do not attempt to use *dosFsVolUnmount()* with volumes mounted using *usrFdConfig()*. This routine does not return the **DOS_VOL_CONFIG** structure required by *dosFsVolUnmount()*. Instead use *ioctl()* with **FIOUNMOUNT**, which accesses the volume information via the file descriptor.

Announcing Disk Changes with Ready-Change

The second method of informing **dosFsLib** that a disk change is taking place is via the "ready-change" mechanism. A change in the disk's ready status is interpreted by **dosFsLib** to indicate that the disk should be remounted during the next I/O operation.

There are three ways to announce a ready-change. First, the *dosFsReadyChange()* routine may be called directly. Second, the *ioctl()* call may be used, with the **FIODISKCHANGE** function code. Finally, the device driver may set the "bd_readyChanged" field in the **BLK_DEV** structure to **TRUE**. This has the same effect as notifying **dosFsLib** directly.

The ready-change mechanism does not provide the ability to flush data structures to the disk. It merely marks the volume as needing remounting. As a result, buffered data (data written to files, directory entries, or FAT changes) may be lost. This may be avoided by synchronizing the disk before asserting ready-change. (The combination of synchronizing and asserting ready-change provides all the functionality of *dosFsVolUnmount()*, except for marking file descriptors as obsolete.)

Since it does not attempt to flush data or to perform other operations that could cause a delay, ready-change may be used in interrupt handlers.

Disks with No Change Notification

If it is not possible for *dosFsVolUnmount()* or *dosFsReadyChange()* to be called each time the disk is changed, the device must be specially identified when it is initialized with the file system. One of the parameters of *dosFsDevInit()* is the address of a **DOS_VOL_CONFIG** structure, which specifies various configuration parameters. **DOS_OPT_CHANGENOWARN** must be set in the **dosvc_options** field of the **DOS_VOL_CONFIG** structure, if the driver and/or application is unable to issue a *dosFsVolUnmount()* call or assert a ready-change when a disk is changed.

This configuration option results in a significant performance disadvantage, because the disk configuration data must be regularly read in from the physical disk, in case the disk has been changed. In addition, setting **DOS_OPT_CHANGENOWARN** also enables auto-sync mode (see below).

Note that for disk change notification, all that is required is that *dosFsVolUnmount()* or *dosFsReadyChange()* be called each time the disk is changed. It is not necessary that either routine be called from the device driver or an interrupt handler. For example, if your application provided a user interface through which an operator could enter a command which would result in a *dosFsVolUnmount()* call before removing the disk, that would be sufficient, and **DOS_OPT_CHANGENOWARN** should not be set. It is important, however, that such a procedure be followed strictly.

Synchronizing Volumes

A disk should be "synchronized" before is is unmounted. To synchronize a disk means to write out all buffered data (files, directories, and the FAT table) that have been modified, so that the disk is "up-to-date." It may or may not be necessary to explicitly synchronize a disk, depending on when (or if) the *dosFsVolUnmount()* call is issued.

When *dosFsVolUnmount()* is called, an attempt will be made to synchronize the device before unmounting. If the disk is still present and writable at the time *dosFsVolUnmount()* is called, the synchronization will take place; there is no need to independently synchronize the disk.

However, if *dosFsVolUnmount()* is called after a disk has been removed, it is obviously too late to synchronize. (In this situation, *dosFsVolUnmount()* discards the buffered data.) Therefore, a separate *ioctl()* call with the **FIOFLUSH** or **FIOSYNC** function should be made before the disk is removed. (This could be done in response to an operator command.)

Auto-Sync Mode

The dosFs file system provides a modified mode of behavior called "auto-sync." This mode is enabled by setting **DOS_OPT_AUTOSYNC** in the **dosvc_options** field of the **DOS_VOL_CONFIG** structure when calling *dosFsDevInit()*. When this option is enabled, modified directory and FAT data is written to the physical device as soon as these structures are altered. (Normally, such changes may not be written out until the involved file is closed.) This results in a performance penalty, but it provides the highest level of data security, since it minimizes the amount of time when directory and FAT data on the disk are not up-to-date.

Auto-sync mode is automatically enabled if the volume does not have disk change notification, i.e., if **DOS_OPT_CHANGENOWARN** is set in the **dosvc_options** field of the **DOS_VOL_CONFIG** structure when *dosFsDevInit()* is called. It may also be desirable for applications where data integrity-- in case of a system crash--is a larger concern than simple disk I/O performance.

CHANGES IN VOLUME CONFIGURATION

Various disk configuration parameters are specified when the dosFs device is first initialized using *dosFsDevInit()*. This data is kept in the volume descriptor (**DOS_VOL_DESC**) for the device. However, it is possible for a disk with different parameters than those defined to be placed in a drive after the device has already been initialized. For such a disk to be usable, the configuration data in the volume descriptor must be modified when a new disk is present.

When a disk is mounted, the boot sector information is read from the disk. This data is used to update the configuration data in the volume descriptor. Note that this will happen the first time the disk is accessed after the volume has been unmounted (using *dosFsVolUnmount()*).

This automatic re-initialization of the configuration data has two important implications:

- (1) Since the values in the volume descriptor are reset when a new volume is mounted, it is possible to omit the dosFs configuration data (by specifying a NULL pointer instead of the address of a **DOS_VOL_CONFIG** structure during *dosFsDevInit()*). The first use of the volume must be with a properly formatted and initialized disk. (Attempting to initialize a disk, using **FIODISKINIT**, before a valid disk has been mounted is fruitless.)
- (2) The volume descriptor data is used when initializing a disk (with **FIODISKINIT**). The **FIODISKINIT** function initializes a disk with the configuration of the most recently mounted disk, regardless of the original specification during *dosFsDevInit()*. Therefore, it is recommended that **FIODISKINIT** be used immediately after

dosFsDevInit(), before any disk has been mounted. (The device should be opened in raw mode; the **FIODISKINIT** function is then performed; and the device is then closed.)

IOCTL FUNCTIONS The dosFs file system supports the following *ioctl()* functions. The functions listed are defined in the header **ioLib.h**. Unless stated otherwise, the file descriptor used for these functions may be any file descriptor which is opened to a file or directory on the volume or to the volume itself.

FIODISKFORMAT

Formats the entire disk with appropriate hardware track and sector marks. No file system is initialized on the disk by this request. Note that this is a driver-provided function:

```
fd = open ("DEV1:", O_WRONLY);  
status = ioctl (fd, FIODISKFORMAT, 0);
```

FIODISKINIT

Initializes a DOS file system on the disk volume. This routine does not format the disk; formatting must be done by the driver. The file descriptor should be obtained by opening the entire volume in raw mode:

```
fd = open ("DEV1:", O_WRONLY);  
status = ioctl (fd, FIODISKINIT, 0);
```

FIODISKCHANGE

Announces a media change. It performs the same function as *dosFsReadyChange()*. This function may be called from interrupt level:

```
status = ioctl (fd, FIODISKCHANGE, 0);
```

FIOUNMOUNT

Unmounts a disk volume. It performs the same function as *dosFsVolUnmount()*. This function must not be called from interrupt level:

```
status = ioctl (fd, FIOUNMOUNT, 0);
```

FIOGETNAME

Gets the file name of the file descriptor and copies it to the buffer *nameBuf*:

```
status = ioctl (fd, FIOGETNAME, &nameBuf );
```

FIORENAME

Renames the file or directory to the string *newname*:

```
status = ioctl (fd, FIORENAME, "newname");
```

FIOSEEK

Sets the current byte offset in the file to the position specified by *newOffset*:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

FIOWHERE

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. It takes no additional argument:

dosFsLib

```
position = ioctl (fd, FIOWHERE, 0);
```

FIOFLUSH

Flushes the file output buffer. It guarantees that any output that has been requested is actually written to the device. If the specified file descriptor was obtained by opening the entire volume (raw mode), this function will flush all buffered file buffers, directories, and the FAT table to the physical device:

```
status = ioctl (fd, FIOFLUSH, 0);
```

FIOSYNC

Performs the same function as FIOFLUSH, and additionally re-reads buffered file data from the disk. This allows file changes made via a different file descriptor to be seen.

FIOTRUNC

Truncates the specified file's length to *newLength* bytes. Any disk clusters which had been allocated to the file but are now unused are returned, and the directory entry for the file is updated to reflect the new length. Only regular files may be truncated; attempts to use FIOTRUNC on directories or the entire volume will return an error. FIOTRUNC may only be used to make files shorter; attempting to specify a *newLength* larger than the current size of the file produces an error (setting errno to S_dosFsLib_INVALID_NUMBER_OF_BYTES).

```
status = ioctl (fd, FIOTRUNC, newLength);
```

FIONREAD

Copies to *unreadCount* the number of unread bytes in the file:

```
status = ioctl (fd, FIONREAD, &unreadCount);
```

FIONFREE

Copies to *freeCount* the amount of free space, in bytes, on the volume:

```
status = ioctl (fd, FIONFREE, &freeCount);
```

FIOMKDIR

Creates a new directory with the name specified as *dirName*:

```
status = ioctl (fd, FIOMKDIR, "dirName");
```

FIORMDIR

Removes the directory whose name is specified as *dirName*:

```
status = ioctl (fd, FIORMDIR, "dirName");
```

FIOLABELGET

Gets the volume label (located in root directory) and copies the string to *labelBuffer*:

```
status = ioctl (fd, FIOLABELGET, &labelBuffer);
```

FIOLABELSET

Sets the volume label to the string specified as *newLabel*. The string may consist of up to eleven ASCII characters:

```
status = ioctl (fd, FIOLABELSET, "newLabel");
```

FIOATTRIBSET

Sets the file attribute byte in the DOS directory entry to the new value *newAttrib*. The file descriptor refers to the file whose entry is to be modified:

```
status = ioctl (fd, FIOATTRIBSET, newAttrib);
```

FIOCONTIG

Allocates contiguous disk space for a file or directory. The number of bytes of requested space is specified in *bytesRequested*. In general, contiguous space should be allocated immediately after the file is created:

```
status = ioctl (fd, FIOCONTIG, bytesRequested);
```

FIONCONTIG

Copies to *maxContigBytes* the size of the largest contiguous free space, in bytes, on the volume:

```
status = ioctl (fd, FIONCONTIG, &maxContigBytes);
```

FIOREADDIR

Reads the next directory entry. The argument *dirStruct* is a DIR directory descriptor. Normally, the *readdir()* routine is used to read a directory, rather than using the **FIOREADDIR** function directly. See **dirLib**.

```
DIR dirStruct;  
fd = open ("directory", O_RDONLY);  
status = ioctl (fd, FIOREADDIR, &dirStruct);
```

FIOFSTATGET

Gets file status information (directory entry data). The argument *statStruct* is a pointer to a stat structure that is filled with data describing the specified file. Normally, the *stat()* or *fstat()* routine is used to obtain file information, rather than using the **FIOFSTATGET** function directly. See **dirLib**.

```
struct stat statStruct;  
fd = open ("file", O_RDONLY);  
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

Any other *ioctl()* function codes are passed to the block device driver for handling.

MEMORY CONSUMPTION

In order to minimize memory fragmentation in the system memory pool, all memory consumed by **dosFsLib** will be contained within a dedicated memory partition. This partition is accessible via the *dosFsMemPartId* global variable.

To display the current amount of memory used by **dosFsLib**, call *show(dosFsMemPartId)*. Please see the manual page for *memPartShow()* for more details.

The following variables may be set *before* **dosFsLib** is initialized to change the behavior of the memory management.

If the **dosFsLib** memory partition is not provided, one will be allocated from the system memory pool. It's size defaults to 8 K, which may be changed via the

dosFsMemPartInitSize global. To provide a memory pool, set *dosFsMemPartId* to a valid **PART_ID** returned from *memPartCreate()*.

The global variable *dosFsMemPartIdOptions* may be modified to change the behavior of error handling for errors in *malloc()* and *free()*. The options default to **MEM_BLOCK_ERROR_LOG_FLAG**, which will log information about errors detected by *free()*. These options only affect operations on the dosFs memory partition.

The private partition will dynamically grow as much as needed, allocating additional memory from the system memory pool, in units no smaller than 1 Kilobyte. This minimum unit size may be adjusted via the *dosFsMemPartGrowSize* global variable.

The maximum size for the dosFs memory partition may be limited via the global variable *dosFsMemPartCap*. Once the cap limit has been reached or surpassed, dosFs will not attempt to allocate more memory from the system memory partition. The default value is -1, which allows uninterrupted use of the system memory partition.

Additional debugging may be enabled via the global boolean *dosFsDebug*. Setting this to 1 will enable verbose debug messages from the dosFs memory manager.

INCLUDE FILES **dosFsLib.h**

SEE ALSO **dosFsLib**, **ioLib**, **iosLib**, **dirLib**, **ramDrv**, *Microsoft MS-DOS Programmer's Reference* (Microsoft Press), *Advanced MS-DOS Programming* (Ray Duncan, Microsoft Press), *VxWorks Programmer's Guide: I/O System, Local File Systems*

ei82596End

NAME **ei82596End** – END style Intel 82596 Ethernet network interface driver

ROUTINES *ei82596EndLoad()* – initialize the driver and device

DESCRIPTION This module implements an Intel 82596 Ethernet network interface driver. This driver is designed to be moderately generic. It operates unmodified across the range of architectures and targets supported by VxWorks. To achieve this, this driver requires some external support routines as well as several target-specific parameters. These parameters (and the mechanisms used to communicate them to the driver) are detailed below.

This driver can run with the device configured in either big-endian or little-endian modes. Error recovery code has been added to deal with some of the known errata in the A0 version of the device. This driver supports up to four individual units per CPU.

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

The driver provides one standard external interface, *ei82596EndLoad()*. As input, this routine takes a string of colon-separated parameters. The parameters should be specified in hexadecimal (optionally preceded by "0x" or a minus sign "-"). The parameter string is parsed using *strtok_r()*, and each parameter is converted from string to binary by a call to:

```
strtoul(parameter, NULL, 16).
```

TARGET-SPECIFIC PARAMETERS

The format of the parameter string is:

```
unit:ivec:sybus:memBase:nTfds:nRfds:offset
```

unit

A convenient holdover from the former model. It is only used in the string name for the driver.

ivec

This is the interrupt vector number of the hardware interrupt generated by this ethernet device. The driver uses *intConnect()* to attach an interrupt handler to this interrupt.

sybus

This parameter tells the device about the system bus. To determine the correct value for a target, see *Intel 32-bit Local Area Network (LAN) Component User's Manual*.

memBase

This parameter specifies the base address of a DMA-able cache-free pre-allocated memory region for use as a memory pool for transmit/receive descriptors, buffers, and other device control structures. If there is no pre-allocated memory available for the driver, this parameter should be -1 (NONE). In which case, the driver calls *cacheDmaAlloc()* to allocate cache-safe memory.

nTfds

This parameter specifies the number of transmit descriptor/buffers to be allocated. If this parameter is zero or -1 (NULL), a default of 32 is used.

nRfds

This parameter specifies the number of receive descriptor/buffers to be allocated. If this parameter is zero or -1 (NULL), a default of 32 is used.

offset

Specifies the memory alignment offset.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires seven external support functions:

sys596IntEnable()

```
void sys596IntEnable (int unit)
```

This routine provides a target-specific interface to enable Ethernet device interrupts for a given device unit.

sys596IntDisable()

void sys596IntDisable (int unit)

This routine provides a target-specific interface to disable Ethernet device interrupts for a given device unit.

sysEnetAddrGet()

STATUS sysEnetAddrGet (int unit, char *enetAdrs)

This routine provides a target-specific interface to access a device Ethernet address. This routine should provide a six-byte Ethernet address in the *enetAdrs* parameter and return OK or ERROR.

sys596Init()

STATUS sys596Init (int unit)

This routine performs any target-specific initialization required before the 82596 is initialized. Typically, it is empty. This routine must return OK or ERROR.

sys596Port()

void sys596Port (int unit, int cmd, UINT32 addr)

This routine provides access to the special port function of the 82596. It delivers the command and address arguments to the port of the specified unit. The driver calls this routine primarily during initialization and, under some conditions, during error recovery procedures.

sys596ChanAtn()

void sys596ChanAtn (int unit)

This routine provides the channel attention signal to the 82596 for the specified *unit*. The driver calls this routine frequently throughout all phases of operation.

sys596IntAck()

void sys596IntAck (int unit)

This routine must perform any required interrupt acknowledgment or clearing. Typically, this involves an operation to some interrupt control hardware. Note that the INT signal from the 82596 behaves in an "edge-triggered" mode. Therefore, this routine typically clears a latch within the control circuitry. The driver calls this routine from the interrupt handler.

SYSTEM RESOURCE USAGE

The driver uses *cacheDmaMalloc()* to allocate memory to share with the 82596. The fixed-size pieces in this area total 160 bytes. The variable-size pieces in this area are affected by the configuration parameters specified in the *eiattach()* call. The size of one RFD (Receive Frame Descriptor) is 1536 bytes. The size of one TFD (Transmit Frame Descriptor) is 1534 bytes. For more on RFDs and TFDs, see the *Intel 82596 User's Manual*.

The 82596 requires ether that this shared memory region is non-cacheable or that the hardware implements bus snooping. The driver cannot maintain cache coherency for the device. This is because fields within the command structures are asynchronously

modified by both the driver and the device, and these fields might share the same cache line.

- TUNING HINTS** The only adjustable parameters are the number of TFDs and RFDs that are created at run-time. These parameters are given to the driver when *eiattach()* is called. There is one TFD and one RFD associated with each transmitted frame and each received frame respectively. For memory-limited applications, decreasing the number of TFDs and RFDs might be a good idea. Increasing the number of TFDs provides no performance benefit after a certain point. Increasing the number of RFDs provides more buffering before packets are dropped. This can be useful if there are tasks running at a higher priority than the net task.
- SEE ALSO** *ifLib*, *Intel 82596 User's Manual*, *Intel 32-bit Local Area Network (LAN) Component User's Manual*

el3c90xEnd

- NAME** **el3c90xEnd** – END network interface driver for 3COM 3C90xB XL
- ROUTINES** *el3c90xEndLoad()* – initialize the driver and device
el3c90xInitParse() – parse the initialization string
- DESCRIPTION** This module implements the device driver for the 3COM EtherLink XI and Fast EtherLink XL PCI network interface cards.
- The 3c90x PCI ethernet controller is inherently little endian because the chip is designed to operate on a PCI bus which is a little endian bus. The software interface to the driver is divided into three parts. The first part is the PCI configuration registers and their set up. This part is done at the BSP level in the various BSPs which use this driver. The second and third part are dealt in the driver. The second part of the interface comprises of the I/O control registers and their programming. The third part of the interface comprises of the descriptors and the buffers.
- This driver is designed to be moderately generic, operating unmodified across the range of architectures and targets supported by VxWorks. To achieve this, the driver must be given several target-specific parameters, and some external support routines must be provided. These target-specific values and the external support routines are described below.
- This driver supports multiple units per CPU. The driver can be configured to support big-endian or little-endian architectures. It contains error recovery code to handle known device errata related to DMA activity.

Big endian processors can be connected to the PCI bus through some controllers which take care of hardware byte swapping. In such cases all the registers which the chip DMA s to have to be swapped and written to, so that when the hardware swaps the accesses, the chip would see them correctly. The chip still has to be programmed to operated in little endian mode as it is on the PCI bus. If the cpu board hardware automatically swaps all the accesses to and from the PCI bus, then input and output byte stream need not be swapped.

The 3c90x series chips use a bus-master DMA interface for transferring packets to and from the controller chip. Some of the old 3c59x cards also supported a bus master mode, however for those chips you could only DMA packets to and from a contiguous memory buffer. For transmission this would mean copying the contents of the queued **M_BLK** chain into a an **M_BLK** cluster and then DMAing the cluster. This extra copy would sort of defeat the purpose of the bus master support for any packet that doesn't fit into a single **M_BLK**. By contrast, the 3c90x cards support a fragment-based bus master mode where **M_BLK** chains can be encapsulated using TX descriptors. This is also called the gather technique, where the fragments in an mBlk chain are directly incorporated into the download transmit descriptor. This avoids any copying of data from the mBlk chain.

NETWORK CARDS SUPPORTED

- 3Com 3c900-TPO 10Mbps/RJ-45
- 3Com 3c900-COMBO 10Mbps/RJ-45,AUI,BNC
- 3Com 3c905-TX 10/100Mbps/RJ-45
- 3Com 3c905-T4 10/100Mbps/RJ-45
- 3Com 3c900B-TPO 10Mbps/RJ-45
- 3Com 3c900B-COMBO 10Mbps/RJ-45,AUI,BNC
- 3Com 3c905B-TX 10/100Mbps/RJ-45
- 3Com 3c905B-FL/FX 10/100Mbps/Fiber-optic
- 3Com 3c980-TX 10/100Mbps server adapter
- Dell Optiplex GX1 on-board 3c918 10/100Mbps/RJ-45

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

The only external interface is the *el3c90xEndLoad()* routine, which expects the *initString* parameter as input. This parameter passes in a colon-delimited string of the format:

```
unit:devMemAddr:devIoAddr:pciMemBase:<vecNum:intLvl:memAdrs:  
memSize:memWidth:flags:buffMultiplier
```

The *el3c90xEndLoad()* function uses *strtok()* to parse the string.

TARGET-SPECIFIC PARAMETERS

unit

A convenient holdover from the former model. This parameter is used only in the string name for the driver.

devMemAddr

This parameter in the memory base address of the device registers in the memory map of the CPU. It indicates to the driver where to find the register set. < This parameter should be equal to NONE if the device does not support memory mapped registers.

devIoAddr

This parameter in the IO base address of the device registers in the IO map of some CPUs. It indicates to the driver where to find the RDP register. If both *devIoAddr* and *devMemAddr* are given then the device chooses *devMemAddr* which is a memory mapped register base address. This parameter should be equal to NONE if the device does not support IO mapped registers.

pciMemBase

This parameter is the base address of the CPU memory as seen from the PCI bus. This parameter is zero for most intel architectures.

vecNum

This parameter is the vector associated with the device interrupt. This driver configures the LANCE device to generate hardware interrupts for various events within the device; thus it contains an interrupt handler routine. The driver calls *intConnect()* to connect its interrupt handler to the interrupt vector generated as a result of the LANCE interrupt.

intLvl

Some targets use additional interrupt controller devices to help organize and service the various interrupt sources. This driver avoids all board-specific knowledge of such devices. During the driver's initialization, the external routine *sysE13c90xIntEnable()* is called to perform any board-specific operations required to allow the servicing of a NIC interrupt. For a description of *sysE13c90xIntEnable()*, see "External Support Requirements" below.

memAdrs

This parameter gives the driver the memory address to carve out its buffers and data structures. If this parameter is specified to be NONE then the driver allocates cache coherent memory for buffers and descriptors from the system pool. The 3C90x NIC is a DMA type of device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the NIC. It assumes that this shared memory is directly available to it without any arbitration or timing concerns.

memSize

This parameter can be used to explicitly limit the amount of shared memory (bytes) this driver will use. The constant NONE can be used to indicate no specific size limitation. This parameter is used only if a specific memory region is provided to the driver.

memWidth

Some target hardware that restricts the shared memory region to a specific location also restricts the access width to this region by the CPU. On these targets, performing an access of an invalid width will cause a bus error.

This parameter can be used to specify the number of bytes of access width to be used by the driver during access to the shared memory. The constant NONE can be used to indicate no restrictions.

Current internal support for this mechanism is not robust; implementation may not work on all targets requiring these restrictions.

flags

This parameter is used for future use, currently its value should be zero.

buffMultiplier

This parameter is used to increase the number of buffers allocated in the driver pool. If this parameter is -1 then a default multiplier of 2 is chosen. With a multiplier of 2 the total number of clusters allocated is 64 which is twice the cumulative number of upload and download descriptors. The device has 16 upload and 16 download descriptors. For example, on choosing the buffer multiplier of 3, the total number of clusters allocated will be 96 $((16 + 16) * 3)$. There are as many cBlks as the number of clusters. The number of mBlks allocated are twice the number of cBlks. By default there are 64 clusters, 64 cBlks and 128 mBlks allocated in the pool for the device. Depending on the load of the system increase the number of clusters allocated by incrementing the buffer multiplier.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires several external support functions, defined as macros:

```
SYS_INT_CONNECT(pDrvCtrl, routine, arg)
SYS_INT_DISCONNECT (pDrvCtrl, routine, arg)
SYS_INT_ENABLE(pDrvCtrl)
SYS_INT_DISABLE(pDrvCtrl)
SYS_OUT_BYTE(pDrvCtrl, reg, data)
SYS_IN_BYTE(pDrvCtrl, reg, data)
SYS_OUT_WORD(pDrvCtrl, reg, data)
SYS_IN_WORD(pDrvCtrl, reg, data)
SYS_OUT_LONG(pDrvCtrl, reg, data)
SYS_IN_LONG(pDrvCtrl, reg, data)
SYS_DELAY (delay)
sysE13c90xIntEnable(pDrvCtrl->intLevel)
sysE13c90xIntDisable(pDrvCtrl->intLevel)
sysDelay (delay)
```

There are default values in the source code for these macros. They presume memory mapped accesses to the device registers and the normal *intConnect()*, and *intEnable()* BSP functions. The first argument to each is the device controller structure. Thus, each

has access back to all the device-specific information. Having the pointer in the macro facilitates the addition of new features to this driver.

The macros `SYS_INT_CONNECT`, `SYS_INT_DISCONNECT`, `SYS_INT_ENABLE`, and `SYS_INT_DISABLE` allow the driver to be customized for BSPs that use special versions of these routines.

The macro `SYS_INT_CONNECT` is used to connect the interrupt handler to the appropriate vector. By default it is the routine `intConnect()`.

The macro `SYS_INT_DISCONNECT` is used to disconnect the interrupt handler prior to unloading the module. By default this is a dummy routine that returns OK.

The macro `SYS_INT_ENABLE` is used to enable the interrupt level for the end device. It is called once during initialization. It calls an external board level routine `sysEl3c90xIntEnable()`.

The macro `SYS_INT_DISABLE` is used to disable the interrupt level for the end device. It is called during stop. It calls an external board level routine `sysEl3c90xIntDisable()`.

The macro `SYS_DELAY` is used for a delay loop. It calls an external board level routine `sysDelay(delay)`. The granularity of delay is one microsecond.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- 24072 bytes in text for a I80486 target
- 112 bytes in the initialized data section (data)
- 0 bytes in the uninitialized data section (BSS)

The driver allocates clusters of size 1536 bytes for receive frames and and transmit frames. There are 16 descriptors in the upload ring and 16 descriptors in the download ring. The buffer multiplier by default is 2, which means that the total number of clusters allocated by default are 64 ((upload descriptors + download descriptors)*2). There are as many cBlks as the number of clusters. The number of mBlks allocated are twice the number of cBlks. By default there are 64 clusters, 64 cBlks and 128 mBlks allocated in the pool for the device. Depending on the load of the system increase the number of clusters allocated by incrementing the buffer multiplier.

INCLUDES `end.h` `endLib.h` `etherMultiLib.h` `el3c90xEnd.h`

SEE ALSO `muxLib`, `endLib`, `netBufLib`, *VxWorks Programmer's Guide: Writing and Enhanced Network Driver*

BIBLIOGRAPHY *3COM 3c90x and 3c90xB NICs Technical reference.*

elt3c509End

| | |
|-----------------------------------|--|
| NAME | elt3c509End – END network interface driver for 3COM 3C509 |
| ROUTINES | <i>elt3c509Load()</i> – initialize the driver and device <i>elt3c509Parse()</i> – parse the init string |
| DESCRIPTION | This module implements the 3COM 3C509 EtherLink III Ethernet network interface driver. This driver is designed to be moderately generic. Thus, it operates unmodified across the range of architectures and targets supported by VxWorks. To achieve this, the driver load routine requires an input string consisting of several target-specific values. The driver also requires some external support routines. These target-specific values and the external support routines are described below. |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | The only external interface is the <i>elt3c509Load()</i> routine, which expects the <i>initString</i> parameter as input. This parameter passes in a colon-delimited string of the format: <i>unit:port:intVector:intLevel:attachementType:nRxFrames</i> The <i>elt3c509Load()</i> function uses <i>strtok()</i> to parse the string. |
| TARGET-SPECIFIC PARAMETERS | <i>unit</i> A convenient holdover from the former model. This parameter is used only in the string name for the driver. <i>intVector</i> Configures the ELT device to generate hardware interrupts for various events within the device. Thus, it contains an interrupt handler routine. The driver calls <i>intConnect()</i> to connect its interrupt handler to the interrupt vector generated as a result of the ELT interrupt. <i>intLevel</i> This parameter is passed to an external support routine, <i>sysEltIntEnable()</i> , which is described below in "External Support Requirements." This routine is called during as part of driver's initialization. It handles any board-specific operations required to allow the servicing of a ELT interrupt on targets that use additional interrupt controller devices to help organize and service the various interrupt sources. This parameter makes it possible for this driver to avoid all board-specific knowledge of such devices. |

attachmentType

This parameter is used to select the transceiver hardware attachment. This is then used by the *elt3c509BoardInit()* routine to activate the selected attachment. *elt3c509BoardInit()* is called as a part of the driver's initialization.

nRxFrames

This parameter is used as number of receive frames by the driver.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires several external support functions, defined as macros:

```

SYS_INT_CONNECT(pDrvCtrl, routine, arg)
SYS_INT_DISCONNECT (pDrvCtrl, routine, arg)
SYS_INT_ENABLE(pDrvCtrl)
SYS_INT_DISABLE(pDrvCtrl)
SYS_OUT_BYTE(pDrvCtrl, reg, data)
SYS_IN_BYTE(pDrvCtrl, reg, data)
SYS_OUT_WORD(pDrvCtrl, reg, data)
SYS_IN_WORD(pDrvCtrl, reg, data)
SYS_OUT_WORD_STRING(pDrvCtrl, reg, pData, len)
SYS_IN_WORD_STRING(pDrvCtrl, reg, pData, len)

sysEltIntEnable(pDrvCtrl->intLevel)
sysEltIntDisable(pDrvCtrl->intLevel)

```

There are default values in the source code for these macros. They presume IO-mapped accesses to the device registers and the normal *intConnect()*, and *intEnable()* BSP functions. The first argument to each is the device controller structure. Thus, each has access back to all the device-specific information. Having the pointer in the macro facilitates the addition of new features to this driver.

The macros *SYS_INT_CONNECT*, *SYS_INT_DISCONNECT*, and *SYS_INT_ENABLE* allow the driver to be customized for BSPs that use special versions of these routines.

The macro *SYS_INT_CONNECT* is used to connect the interrupt handler to the appropriate vector. By default it is the routine *intConnect()*.

The macro *SYS_INT_DISCONNECT* is used to disconnect the interrupt handler prior to unloading the module. By default this is a dummy routine that returns OK.

The macro *SYS_INT_ENABLE* is used to enable the interrupt level for the end device. It is called once during initialization. It calls an external board level routine *sysEltIntEnable()*.

The macro *SYS_INT_DISABLE* is used to disable the interrupt level for the end device. It is called during stop. It calls an external board level routine *sysEltIntDisable()*.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

endLib

- one interrupt vector
- 9720 bytes of text
- 88 bytes in the initialized data section (data)
- 0 bytes of bss

The driver requires 1520 bytes of preallocation for Transmit Buffer and 1520*nRxFrames of receive buffers. The default value of nRxFrames is 64 therefore total pre-allocation is (64 + 1)*1520.

| | |
|---------------------|---|
| TUNING HINTS | nRxFrames parameter can be used for tuning no of receive frames to be used for handling packet receive. More no. of these could help receiving more loaning in case of massive reception. |
| INCLUDES | end.h endLib.h etherMultiLib.h elt3c509End.h |
| SEE ALSO | muxLib, endLib <i>Writing and Enhanced Network Driver</i> |

endLib

| | |
|--------------------|---|
| NAME | endLib – support library for END-based drivers |
| ROUTINES | <i>mib2Init()</i> – initialize a MIB-II structure <i>mib2ErrorAdd()</i> – change a MIB-II error count <i>endObjInit()</i> – initialize an END_OBJ structure <i>endObjFlagSet()</i> – set the flags member of an END_OBJ structure <i>endEtherAddressForm()</i> – form an Ethernet address into a packet <i>endEtherPacketDataGet()</i> – return the beginning of the packet data <i>endEtherPacketAddrGet()</i> – locate the addresses in a packet |
| DESCRIPTION | This library contains support routines for Enhanced Network Drivers. These routines are common to ALL ENDS. Specialized routines should only appear in the drivers themselves. |

envLib

| | |
|-----------------|--|
| NAME | envLib – environment variable library |
| ROUTINES | <i>envLibInit()</i> – initialize environment variable facility <i>envPrivateCreate()</i> – create a private environment |

envPrivateDestroy() – destroy a private environment
putenv() – set an environment variable
getenv() – get an environment variable (ANSI)
envShow() – display the environment for a task

DESCRIPTION

This library provides a UNIX-compatible environment variable facility. Environment variables are created or modified with a call to *putenv()*:

```
putenv ("variableName=value");
```

The value of a variable may be retrieved with a call to *getenv()*, which returns a pointer to the value string.

Tasks may share a common set of environment variables, or they may optionally create their own private environments, either automatically when the task create hook is installed, or by an explicit call to *envPrivateCreate()*. The task must be spawned with the `VX_PRIVATE_ENV` option set to receive a private set of environment variables. Private environments created by the task creation hook inherit the values of the environment of the task that called *taskSpawn()* (since task create hooks run in the context of the calling task).

INCLUDE FILES

envLib.h

SEE ALSO

UNIX BSD 4.3 manual entry for **environ(5V)**, * *American National Standard for Information Systems – C Programming Language – C, ANSI X3.159-1989: General Utilities (stdlib.h)*

errnoLib

NAME

errnoLib – error status library

ROUTINES

errnoGet() – get the error status value of the calling task
errnoOfTaskGet() – get the error status value of a specified task
errnoSet() – set the error status value of the calling task
errnoOfTaskSet() – set the error status value of a specified task

DESCRIPTION

This library contains routines for setting and examining the error status values of tasks and interrupts. Most VxWorks functions return ERROR when they detect an error, or NULL in the case of functions returning pointers. In addition, they set an error status that elaborates the nature of the error.

This facility is compatible with the UNIX error status mechanism in which error status values are set in the global variable **errno**. However, in VxWorks there are many task and interrupt contexts that share common memory space and therefore conflict in their use of this global variable. VxWorks resolves this in two ways:

- (1) For tasks, VxWorks maintains the **errno** value for each context separately, and saves and restores the value of **errno** with every context switch. The value of **errno** for a non-executing task is stored in the task's TCB. Thus, regardless of task context, code can always reference or modify **errno** directly.
- (2) For interrupt service routines, VxWorks saves and restores **errno** on the interrupt stack as part of the interrupt enter and exit code provided automatically with the **intConnect()** facility. Thus, interrupt service routines can also reference or modify **errno** directly.

The **errno** facility is used throughout VxWorks for error reporting. In situations where a lower-level routine has generated an error, by convention, higher-level routines propagate the same error status, leaving **errno** with the value set at the deepest level. Developers are encouraged to use the same mechanism for application modules where appropriate.

ERROR STATUS VALUES

An error status is a 4-byte integer. By convention, the most significant two bytes are the module number, which indicates the module in which the error occurred. The lower two bytes indicate the specific error within that module. Module number 0 is reserved for UNIX error numbers so that values from the UNIX **errno.h** header file can be set and tested without modification. Module numbers 1-500 decimal are reserved for VxWorks modules. These are defined in **vwModNum.h**. All other module numbers are available to applications.

PRINTING ERROR STATUS VALUES

VxWorks can include a special symbol table called **statSymTbl** which **printErrno()** uses to print human-readable error messages.

This table is created with the tool **makeStatTbl**, found in **host/hostOs/bin**. This tool reads all the **.h** files in a specified directory and generates a C-language file, which generates a symbol table when compiled. Each symbol consists of an error status value and its definition, which was obtained from the header file.

For example, suppose the header file **target/h/myFile.h** contains the line:

```
#define S_myFile_ERROR_TOO_MANY_COOKS    0x230003
```

The table **statSymTbl** is created by first running:

On UNIX:

```
makeStatTbl target/h > statTbl.c
```

On Windows:

```
makeStatTbl target/h
```

This creates a file **statTbl.c** in the current directory, which, when compiled, generates **statSymTbl**. The table is then linked in with VxWorks. Normally, these steps are performed automatically by the makefile in **target/src/usr**.

If the user now types from the VxWorks shell:

```
-> printErrno 0x230003
```

The `printErrno()` routine would respond:

```
S_myFile_ERROR_TOO_MANY_COOKS
```

The `makeStatTbl` tool looks for error status lines of the form:

```
#define S_xxx n
```

where `xxx` is any string, and `n` is any number. All VxWorks status lines are of the form:

```
#define S_thisFile_MEANINGFUL_ERROR_MESSAGE 0xnnnn
```

where `thisFile` is the name of the module.

This facility is available to the user by adding header files with status lines of the appropriate forms and remaking VxWorks.

INCLUDE FILES

The file `vwModNum.h` contains the module numbers for every VxWorks module. The include file for each module contains the error numbers which that module can generate.

SEE ALSO

`printErrno()`, `makeStatTbl`, *VxWorks Programmer's Guide: Basic OS*

etherLib

NAME

etherLib – Ethernet raw I/O routines and hooks

ROUTINES

`etherOutput()` – send a packet on an Ethernet interface
`etherInputHookAdd()` – add a routine to receive all Ethernet input packets
`etherInputHookDelete()` – delete a network interface input hook routine
`etherOutputHookAdd()` – add a routine to receive all Ethernet output packets
`etherOutputHookDelete()` – delete a network interface output hook routine
`etherAddrResolve()` – resolve an Ethernet address for a specified Internet address
`etherTypeGet()` – get the type from an ethernet packet

DESCRIPTION

This library provides utilities that give direct access to Ethernet packets. Raw packets can be output directly to an interface using `etherOutput()`. Incoming and outgoing packets can be examined or processed using the hooks `etherInputHookAdd()` and `etherOutputHookAdd()`. The input hook can be used to receive raw packets that are not part of any of the supported network protocols. The input and output hooks can also be used to build network monitoring and testing tools.

Normally, the network should be accessed through the higher-level socket interface provided in **sockLib**. The routines in **etherLib** should rarely, if ever, be necessary for applications.

CAVEAT

The following VxWorks network drivers support both the input-hook and output-hook routines:

- if_cpm** – Motorola MC68EN360 QUICC network interface driver
- if_eex** – Intel EtherExpress 16
- if_ei** – Intel 82596 ethernet driver
- if_elc** – SMC 8013WC Ethernet driver
- if_elt** – 3Com 3C509 Ethernet driver
- if_ene** – Novell/Eagle NE2000 network driver
- if_fn** – Fujitsu MB86960 NICE Ethernet driver
- if_ln** – Advanced Micro Devices Am7990 LANCE Ethernet driver
- if_sm** – shared memory backplane network interface driver
- if_sn** – National Semiconductor DP83932B SONIC Ethernet driver
- if_ultra** – SMC Elite Ultra Ethernet network interface driver

- if_gn** – generic MUX interface layer

The following drivers support only the input-hook routines:

- if_nic** – National Semiconductor SNIC Chip (for HKV30)
- if_sl** – Serial Line IP (SLIP) network interface driver

The following drivers support only the output-hook routines:

- if_ulip** – network interface driver for User Level IP (VxSim)

The following drivers do not support either the input-hook or output-hook routines:

- if_loop** – software loopback network interface driver

INCLUDE FILES

etherLib.h

SEE ALSO

VxWorks Programmer's Guide: Network

etherMultiLib

| | |
|----------------------|--|
| NAME | etherMultiLib – a library to handle Ethernet multicast addresses |
| ROUTINES | <i>etherMultiAdd()</i> – add multicast address to a multicast address list <i>etherMultiDel()</i> – delete an Ethernet multicast address record <i>etherMultiGet()</i> – retrieve a table of multicast addresses from a driver |
| DESCRIPTION | This library manages a list of multicast addresses for network drivers. This abstracts the management of these drivers into a device independent library. |
| INCLUDE FILES | string.h, errno.h, netinet/in.h, net/if.h, lstLib.h, etherMultiLib.h |
| SEE ALSO | etherMultiLib |

evbNs16550Sio

| | |
|------------------------|---|
| NAME | evbNs16550Sio – NS16550 serial driver for the IBM PPC403GA evaluation |
| ROUTINES | <i>evbNs16550HrdInit()</i> – initialize the NS 16550 chip <i>evbNs16550Int()</i> – handle a receiver/transmitter interrupt for the NS 16550 chip |
| DESCRIPTION | This is the driver for the National NS 16550 UART Chip used on the IBM PPC403GA evaluation board. It uses the SCCs in asynchronous mode only. |
| USAGE | An EVBNs16550_CHAN structure is used to describe the chip. The BSP's <i>sysHwInit()</i> routine typically calls <i>sysSerialHwInit()</i> which initializes all the register values in the EVBNs16550_CHAN structure (except the SIO_DRV_FUNCS) before calling <i>evbNs16550HrdInit()</i> . The BSP's <i>sysHwInit2()</i> routine typically calls <i>sysSerialHwInit2()</i> which connects the chip interrupt handler <i>evbNs16550Int()</i> via <i>intConnect()</i> . |
| IOCTL FUNCTIONS | This driver responds to the same <i>ioctl()</i> codes as other serial drivers; for more information, see sioLib.h . |
| INCLUDE FILES | drv/sio/evbNs16550Sio.h |
| SEE ALSO | evbNs16550Sio |

excArchLib

| | |
|----------------------|--|
| NAME | excArchLib – architecture-specific exception-handling facilities |
| ROUTINES | <i>excVecInit()</i> – initialize the exception/interrupt vectors <i>excConnect()</i> – connect a C routine to an exception vector (PowerPC) <i>excIntConnect()</i> – connect a C routine to an asynchronous exception vector (PowerPC, ARM) <i>excCrtConnect()</i> – connect a C routine to a critical exception vector (PowerPC 403) <i>excIntCrtConnect()</i> – connect a C routine to a critical interrupt vector (PowerPC 403) <i>excVecSet()</i> – set a CPU exception vector (PowerPC, ARM) <i>excVecGet()</i> – get a CPU exception vector (PowerPC, ARM) |
| DESCRIPTION | This library contains exception-handling facilities that are architecture dependent. For information about generic (architecture-independent) exception-handling, see the manual entry for excLib . |
| INCLUDE FILES | excLib.h |
| SEE ALSO | excLib , dbgLib , sigLib , intLib |

excLib

| | |
|-----------------------|---|
| NAME | excLib – generic exception handling facilities |
| ROUTINES | <i>excInit()</i> – initialize the exception handling package <i>excHookAdd()</i> – specify a routine to be called with exceptions <i>excTask()</i> – handle task-level exceptions |
| DESCRIPTION | This library provides generic initialization facilities for handling exceptions. It safely traps and reports exceptions caused by program errors in VxWorks tasks, and it reports occurrences of interrupts that are explicitly connected to other handlers. For information about architecture-dependent exception handling facilities, see the manual entry for excArchLib . |
| INITIALIZATION | Initialization of excLib facilities occurs in two steps. First, the routine <i>excVecInit()</i> is called to set all vectors to the default handlers for an architecture provided by the corresponding architecture exception handling library. Since this does not involve VxWorks' kernel facilities, it is usually done early in the system start-up routine <i>usrInit()</i> in the library usrConfig.c with interrupts disabled. |

The rest of this package is initialized by calling *excInit()*, which spawns the exception support task, *excTask()*, and creates the message queues used to communicate with it.

Exceptions or uninitialized interrupts that occur after the vectors have been initialized by *excVecInit()*, but before *excInit()* is called, cause a trap to the ROM monitor.

NORMAL EXCEPTION HANDLING

When a program error generates an exception (such as divide by zero, or a bus or address error), the task that was executing when the error occurred is suspended, and a description of the exception is displayed on standard output. The VxWorks kernel and other system tasks continue uninterrupted. The suspended task can be examined with the usual VxWorks routines, including *ti()* for task information and *tt()* for a stack trace. It may be possible to fix the task and resume execution with *tr()*. However, tasks aborted in this way are often unsalvageable and can be deleted with *td()*.

When an interrupt that is not connected to a handler occurs, the default handler provided by the architecture-specific module displays a description of the interrupt on standard output.

ADDITIONAL EXCEPTION HANDLING ROUTINE

The *excHookAdd()* routine adds a routine that will be called when a hardware exception occurs. This routine is called at the end of normal exception handling.

TASK-LEVEL SUPPORT

The *excInit()* routine spawns *excTask()*, which performs special exception handling functions that need to be done at task level. Do not suspend, delete, or change the priority of this task.

DBGLIB The facilities of **excLib**, including *excTask()*, are used by **dbgLib** to support breakpoints, single-stepping, and additional exception handling functions.

SIGLIB A higher-level, UNIX-compatible interface for hardware and software exceptions is provided by **sigLib**. If *sigvec()* is used to initialize the appropriate hardware exception/interrupt (e.g., BUS ERROR == SIGSEGV), **excLib** will use the signal mechanism instead.

INCLUDE FILES **excLib.h**

SEE ALSO **dbgLib, sigLib, intLib**

fei82557End

| | |
|---------------------------|---|
| NAME | fei82557End – END style Intel 82557 Ethernet network interface driver |
| ROUTINES | <i>fei82557EndLoad()</i> – initialize the driver and device |
| DESCRIPTION | <p>This module implements an Intel 82557 Ethernet network interface driver. This is a fast Ethernet PCI bus controller, IEEE 802.3 10Base-T and 100Base-T compatible. It also features a glueless 32-bit PCI bus master interface, fully compliant with PCI Spec version 2.1. An interface to MII compliant physical layer devices is built-in in the card. The 82557 Ethernet PCI bus controller also includes Flash support up to 1 MByte and EEPROM support, although these features are not dealt with in the driver.</p> <p>The 82557 establishes a shared memory communication system with the CPU, which is divided into three parts: the Control/Status Registers (CSR), the Command Block List (CBL) and the Receive Frame Area (RFA). The CSR is on chip and is either accessible with I/O or memory cycles, whereas the other structures reside on the host.</p> <p>The CSR is the main means of communication between the device and the host, meaning that the latter issues commands through these registers while the chip posts status changes in it, occurred as a result of those commands. Pointers to both the CBL and RFA are also stored in the CSR.</p> <p>The CBL consists of a linked list of frame descriptors through which individual action commands can be performed. These may be transmit commands as well as non-transmit commands, e.g. Configure or Multicast setup commands. While the CBL list may function in two different modes, only the simplified memory mode is implemented in the driver.</p> <p>The RFA is a linked list of receive frame descriptors. Only support for the simplified memory mode is granted. In this model, the data buffer immediately follows the related frame descriptor.</p> <p>The driver is designed to be moderately generic, operating unmodified across the range of architectures and targets supported by VxWorks. To achieve this, this driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below.</p> |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | <p>The driver provides the standard external interface, <i>fei82557EndLoad()</i>, which takes a string of colon separated parameters. The parameters should be specified in hexadecimal, optionally preceded by "0x" or a minus sign "-".</p> <p>The parameter string is parsed using <i>strtok_r()</i> and each parameter is converted from a string representation to binary by a call to <i>strtoul(parameter, NULL, 16)</i>.</p> |

The format of the parameter string is:

"memBase:memSize:nTfds:nRfds:flags"

In addition, the two global variables **feiEndIntConnect** and **feiEndIntDisconnect** specify respectively the interrupt connect routine and the interrupt disconnect routine to be used depending on the BSP. The former defaults to *intConnect()* and the user can override this to use any other interrupt connect routine (say *pciIntConnect()*) in *sysHwInit()* or any device specific initialization routine called in *sysHwInit()*. Likewise, the latter is set by default to NULL, but it may be overridden in the BSP in the same way.

TARGET-SPECIFIC PARAMETERS

memBase

This parameter is passed to the driver via *fei82557EndLoad()*.

The Intel 82557 device is a DMA-type device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the 82557.

This parameter can be used to specify an explicit memory region for use by the 82557. This should be done on targets that restrict the 82557 to a particular memory region. The constant **NONE** can be used to indicate that there are no memory limitations, in which case the driver will allocate cache safe memory for its use using *cacheDmaAlloc()*.

memSize

The memory size parameter specifies the size of the pre-allocated memory region. If memory base is specified as **NONE** (-1), the driver ignores this parameter. Otherwise, the driver checks the size of the provided memory region is adequate with respect to the given number of Command Frame Descriptor and Receive Frame Descriptor.

nTfds

This parameter specifies the number of transmit descriptor/buffers to be allocated. If this parameter is less than two, a default of 32 is used.

nRfds

This parameter specifies the number of receive descriptor/buffers to be allocated. If this parameter is less than two, a default of 32 is used.

flags

User flags may control the run-time characteristics of the Ethernet chip. Not implemented.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires one external support function:

```
STATUS sys557Init (int unit, FEI_BOARD_INFO *pBoard)
```

fioLib

This routine performs any target-specific initialization required before the 82557 device is initialized by the driver. The driver calls this routine every time it wants to [re]initialize the device. This routine returns OK, or ERROR if it fails.

SYSTEM RESOURCE USAGE

The driver calls *cacheDmaMalloc()* to allocate memory to share with the 82557. The size of this area is affected by the configuration parameters specified by *fei82557EndLoad()*.

Either the shared memory region must be non-cacheable, or else the hardware must implement bus snooping. The driver cannot maintain cache coherency for the device because fields within the command structures are asynchronously modified by both the driver and the device, and these fields may share the same cache line.

TUNING HINTS

The only adjustable parameters are the number of TFDs and RFDs that will be created at run-time. These parameters are given to the driver when *fei82557EndLoad()* is called. There is one TFD and one RFD associated with each transmitted frame and each received frame respectively. For memory-limited applications, decreasing the number of TFDs and RFDs may be desirable. Increasing the number of TFDs will provide no performance benefit after a certain point. Increasing the number of RFDs will provide more buffering before packets are dropped. This can be useful if there are tasks running at a higher priority than the net task.

SEE ALSO

ifLib, *Intel 82557 User's Manual*, *Intel 32-bit Local Area Network (LAN) Component User's Manual*

fioLib

NAME

fioLib – formatted I/O library

ROUTINES

fioLibInit() – initialize the formatted I/O support library
printf() – write a formatted string to the standard output stream (ANSI)
printErr() – write a formatted string to the standard error stream
fdprintf() – write a formatted string to a file descriptor
sprintf() – write a formatted string to a buffer (ANSI)
vprintf() – write a string formatted with a variable argument list to standard output (ANSI)
vfprintf() – write a string formatted with a variable argument list to a file descriptor
vsprintf() – write a string formatted with a variable argument list to a buffer (ANSI)
fioFormatV() – convert a format string
fioRead() – read a buffer
fioRdString() – read a string from a file
sscanf() – read and convert characters from an ASCII string (ANSI)

| | |
|----------------------|---|
| DESCRIPTION | <p>This library provides the basic formatting and scanning I/O functions. It includes some routines from the ANSI-compliant <i>printf()</i>/<i>scanf()</i> family of routines. It also includes several utility routines.</p> <p>If the floating-point format specifications e, E, f, g, and G are to be used with these routines, the routine <i>floatInit()</i> must be called first. If the configuration macro <code>INCLUDE_FLOATING_POINT</code> is defined, <i>floatInit()</i> is called by the root task, <i>usrRoot()</i>, in <code>usrConfig.c</code>.</p> <p>These routines do not use the buffered I/O facilities provided by the standard I/O facility. Thus, they can be invoked even if the standard I/O package has not been included. This includes <i>printf()</i>, which in most UNIX systems is part of the buffered standard I/O facilities. Because <i>printf()</i> is so commonly used, it has been implemented as an unbuffered I/O function. This allows minimal formatted I/O to be achieved without the overhead of the entire standard I/O package. For more information, see the manual entry for <code>ansiStdio</code>.</p> |
| INCLUDE FILES | <code>floatLib.h</code> , <code>stdio.h</code> |
| SEE ALSO | <code>ansiStdio</code> , <code>floatLib</code> , <i>VxWorks Programmer's Guide: I/O System</i> |

floatLib

| | |
|----------------------|--|
| NAME | <code>floatLib</code> – floating-point formatting and scanning library |
| ROUTINES | <i>floatInit()</i> – initialize floating-point I/O support |
| DESCRIPTION | <p>This library provides the floating-point I/O formatting and scanning support routines.</p> <p>The floating-point formatting and scanning support routines are not directly callable; they are connected to call-outs in the <i>printf()</i>/<i>scanf()</i> family of functions in <code>floatLib</code>. This is done dynamically by the routine <i>floatInit()</i>, which is called by the root task, <i>usrRoot()</i>, in <code>usrConfig.c</code> when the configuration macro <code>INCLUDE_FLOATING_POINT</code> is defined. If this option is omitted (i.e., <i>floatInit()</i> is not called), floating-point format specifications in <i>printf()</i> and <i>scanf()</i> are not supported.</p> |
| INCLUDE FILES | <code>math.h</code> |
| SEE ALSO | <code>floatLib</code> |

fppArchLib

| | |
|-----------------------|---|
| NAME | fppArchLib – architecture-dependent floating-point coprocessor support |
| ROUTINES | <i>fppSave()</i> – save the floating-point coprocessor context <i>fppRestore()</i> – restore the floating-point coprocessor context <i>fppProbe()</i> – probe for the presence of a floating-point coprocessor <i>fppTaskRegsGet()</i> – get the floating-point registers from a task TCB <i>fppTaskRegsSet()</i> – set the floating-point registers of a task |
| DESCRIPTION | <p>This library contains architecture-dependent routines to support the floating-point coprocessor. The routines <i>fppSave()</i> and <i>fppRestore()</i> save and restore all the task floating-point context information. The routine <i>fppProbe()</i> checks for the presence of the floating-point coprocessor. The routines <i>fppTaskRegsSet()</i> and <i>fppTaskRegsGet()</i> inspect and set coprocessor registers on a per-task basis.</p> <p>With the exception of <i>fppProbe()</i>, the higher-level facilities in dbgLib and usrLib should be used instead of these routines. For information about architecture-independent access mechanisms, see the manual entry for fppLib.</p> |
| INITIALIZATION | <p>To activate floating-point support, <i>fppInit()</i> must be called before any tasks using the coprocessor are spawned. This is done by the root task, <i>usrRoot()</i>, in usrConfig.c. See the manual entry for fppLib.</p> <p>NOTE I386/I486 On this architecture, VxWorks disables the six FPU exceptions that can send an IRQ to the CPU.</p> |
| NOTE ARM | This architecture does not currently support floating-point coprocessors. |
| INCLUDE FILES | fppLib.h |
| SEE ALSO | fppLib , <i>intConnect()</i> , <i>Motorola MC68881/882 Floating-Point Coprocessor User's Manual</i> , <i>SPARC Architecture Manual</i> , <i>Intel 80960SA/SB Reference Manual</i> , <i>Intel 80960KB Programmer's Reference Manual</i> , <i>Intel 387 DX User's Manual</i> , Gerry Kane and Joe Heinrich: <i>MIPS RISC Architecture Manual</i> |

fppLib

| | |
|--------------------------|---|
| NAME | fppLib – floating-point coprocessor support library |
| ROUTINES | <i>fppInit()</i> – initialize floating-point coprocessor support |
| DESCRIPTION | <p>This library provides a general interface to the floating-point coprocessor. To activate floating-point support, <i>fppInit()</i> must be called before any tasks using the coprocessor are spawned. This is done automatically by the root task, <i>usrRoot()</i>, in usrConfig.c when the configuration macro INCLUDE_HW_FP is defined.</p> <p>For information about architecture-dependent floating-point routines, see the manual entry for fppArchLib.</p> <p>The <i>fppShow()</i> routine displays coprocessor registers on a per-task basis. For information on this facility, see the manual entries for fppShow and <i>fppShow()</i>.</p> |
| VX_FP_TASK OPTION | <p>Saving and restoring floating-point registers adds to the context switch time of a task. Therefore, floating-point registers are not saved and restored for every task. Only those tasks spawned with the task option VX_FP_TASK will have floating-point registers saved and restored.</p> |
| NOTE | If a task does any floating-point operations, it must be spawned with VX_FP_TASK . |
| INTERRUPT LEVEL | Floating-point registers are not saved and restored for interrupt service routines connected with <i>intConnect()</i> . However, if necessary, an interrupt service routine can save and restore floating-point registers by calling routines in fppArchLib . |
| INCLUDE FILES | fppLib.h |
| SEE ALSO | fppArchLib , fppShow , <i>intConnect()</i> , <i>VxWorks Programmer's Guide: Basic OS</i> |

fppShow

| | |
|----------------------|--|
| NAME | fppShow – floating-point show routines |
| ROUTINES | <i>fppShowInit()</i> – initialize the floating-point show facility <i>fppTaskRegsShow()</i> – print the contents of a task’s floating-point registers |
| DESCRIPTION | <p>This library provides the routines necessary to show a task’s optional floating-point context. To use this facility, it must first be installed using <i>fppShowInit()</i>, which is called automatically when the floating-point show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none">– If you use configuration header files, define INCLUDE_SHOW_ROUTINES in config.h.– If you use the Tornado project facility, select INCLUDE_HW_FP_SHOW. <p>This library enhances task information routines, such as <i>ti()</i>, to display the floating-point context.</p> |
| INCLUDE FILES | fppLib.h |
| SEE ALSO | fppLib |

ftpdLib

| | |
|-----------------------|---|
| NAME | ftpdLib – File Transfer Protocol (FTP) server |
| ROUTINES | <i>ftpdInit()</i> – initialize the FTP server task <i>ftpdDelete()</i> – terminate the FTP server task |
| DESCRIPTION | <p>This library implements the server side of the File Transfer Protocol (FTP), which provides remote access to the file systems available on a target. The protocol is defined in RFC 959. This implementation supports all commands required by that specification, as well as several additional commands.</p> |
| USER INTERFACE | <p>During system startup, the <i>ftpdInit()</i> routine creates a control connection at the predefined FTP server port which is monitored by the primary FTP task. Each FTP session established is handled by a secondary server task created as necessary. The server accepts the following commands:</p> <ul style="list-style-type: none">HELP – List supported commands.USER – Verify user name. |

| | |
|------|---|
| PASS | - Verify password for the user. |
| QUIT | - Quit the session. |
| LIST | - List out contents of a directory. |
| NLST | - List directory contents using a concise format. |
| RETR | - Retrieve a file. |
| STOR | - Store a file. |
| CWD | - Change working directory. |
| TYPE | - Change the data representation type. |
| PORT | - Change the port number. |
| PWD | - Get the name of current working directory. |
| STRU | - Change file structure settings. |
| MODE | - Change file transfer mode. |
| ALLO | - Reserver sufficient storage. |
| ACCT | - Identify the user's account. |
| PASV | - Make the server listen on a port for data connection. |
| NOOP | - Do nothing. |
| DELE | - Delete a file |

The *ftpdDelete()* routine will disable the FTP server until restarted. It reclaims all system resources used by the server tasks and cleanly terminates all active sessions.

INCLUDE FILES *ftpdLib.h*

SEE ALSO *ftpLib, netDrv, RFC-959 File Transfer Protocol*

ftpLib

NAME *ftpLib* – File Transfer Protocol (FTP) library

ROUTINES *ftpCommand()* – send an FTP command and get the reply
ftpXfer() – initiate a transfer via FTP
ftpReplyGet() – get an FTP command reply
ftpHookup() – get a control connection to the FTP server on a specified host
ftpLogin() – log in to a remote FTP server
ftpDataConnInit() – initialize an FTP data connection
ftpDataConnGet() – get a completed FTP data connection
ftpLs() – list directory contents via FTP

DESCRIPTION This library provides facilities for transferring files to and from a host via File Transfer Protocol (FTP). This library implements only the "client" side of the FTP facilities.

FTP IN VXWORKS VxWorks provides an I/O driver, **netDrv**, that allows transparent access to remote files via standard I/O system calls. The FTP facilities of **ftpLib** are primarily used by **netDrv** to access remote files. Thus for most purposes, it is not necessary to be familiar with **ftpLib**.

HIGH-LEVEL INTERFACE

The routines *ftpXfer()* and *ftpReplyGet()* provide the highest level of direct interface to FTP. The routine *ftpXfer()* connects to a specified remote FTP server, logs in under a specified user name, and initiates a specified data transfer command. The routine *ftpReplyGet()* receives control reply messages sent by the remote FTP server in response to the commands sent.

LOW-LEVEL INTERFACE

The routines *ftpHookup()*, *ftpLogin()*, *ftpDataConnInit()*, *ftpDataConnGet()*, and *ftpCommand()* provide the primitives necessary to create and use control and data connections to remote FTP servers. The following example shows how to use these low-level routines. It implements roughly the same function as *ftpXfer()*.

```
char *host, *user, *passwd, *acct, *dirname, *filename;
int ctrlSock = ERROR;
int dataSock = ERROR;
if (((ctrlSock = ftpHookup (host)) == ERROR) ||
    (ftpLogin (ctrlSock, user, passwd, acct) == ERROR) ||
    (ftpCommand (ctrlSock, "TYPE I", 0, 0, 0, 0, 0, 0) != FTP_COMPLETE) ||
    (ftpCommand (ctrlSock, "CWD %s", dirname, 0, 0, 0, 0, 0) != FTP_COMPLETE) ||
    ((dataSock = ftpDataConnInit (ctrlSock)) == ERROR) ||
    (ftpCommand (ctrlSock, "RETR %s", filename, 0, 0, 0, 0, 0) != FTP_PRELIM) ||
    ((dataSock = ftpDataConnGet (dataSock)) == ERROR))
{
    /* an error occurred; close any open sockets and return */
    if (ctrlSock != ERROR)
        close (ctrlSock);
    if (dataSock != ERROR)
        close (dataSock);
    return (ERROR);
}
```

INCLUDE FILES ftpLib.h

SEE ALSO netDrv

hostLib

| | |
|----------------------|---|
| NAME | hostLib – host table subroutine library |
| ROUTINES | <i>hostTblInit()</i> – initialize the network host table <i>hostAdd()</i> – add a host to the host table <i>hostDelete()</i> – delete a host from the host table <i>hostGetByName()</i> – look up a host in the host table by its name <i>hostGetByAddr()</i> – look up a host in the host table by its Internet address <i>sethostname()</i> – set the symbolic name of this machine <i>gethostname()</i> – get the symbolic name of this machine |
| DESCRIPTION | <p>This library provides routines to store and access the network host database. The host table contains information regarding the known hosts on the local network. The host table (displayed with <i>hostShow()</i>) contains the Internet address, the official host name, and aliases.</p> <p>By convention, network addresses are specified in dotted (".") decimal notation. The library inetLib contains Internet address manipulation routines. Host names and aliases may contain any printable character.</p> <p>Before any of the routines in this module can be used, the library must be initialized by <i>hostTblInit()</i>. This is done automatically if the configuration macro INCLUDE_NET_INIT is defined.</p> |
| INCLUDE FILES | hostLib.h |
| SEE ALSO | inetLib , <i>VxWorks Programmer's Guide: Network</i> |

i8250Sio

| | |
|--------------------|---|
| NAME | i8250Sio – I8250 serial driver |
| ROUTINES | <i>i8250HrdInit()</i> – initialize the chip <i>i8250Int()</i> – handle a receiver/transmitter interrupt |
| DESCRIPTION | This is the driver for the Intel 8250 UART Chip used on the PC 386. It uses the SCCs in asynchronous mode only. |
| USAGE | An I8250_CHAN structure is used to describe the chip. The BSP's <i>sysHwInit()</i> routine typically calls <i>sysSerialHwInit()</i> which initializes all the register values in the |

icmpShow

i8250_CHAN structure (except the **SIO_DRV_FUNCS**) before calling **i8250HrdInit()**. The BSP's **sysHwInit2()** routine typically calls **sysSerialHwInit2()** which connects the chips interrupt handler (**i8250Int**) via **intConnect()**.

IOCTL FUNCTIONS This driver responds to all the same **ioctl()** codes as a normal serial driver; for more information, see the comments in **sioLib.h**. As initialized, the available baud rates are 110, 300, 600, 1200, 2400, 4800, 9600, 19200, and 38400.

This driver handles setting of hardware options such as parity(odd, even) and number of data bits(5, 6, 7, 8). Hardware flow control is provided with the handshakes RTS/CTS. The function HUPCL(hang up on last close) is available.

INCLUDE FILES **drv/sio/i8250Sio.h**

icmpShow

NAME **icmpShow** – ICMP Information display routines

ROUTINES **icmpShowInit()** – initialize ICMP show routines
icmpstatShow() – display statistics for ICMP

DESCRIPTION This library provides routines to show ICMP related statistics.

Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:

- *TCP/IP Illustrated Volume II, The Implementation*, by Richard Stevens
- *The Design and Implementation of the 4.4 BSD UNIX Operating System*, by Leffler, McKusick, Karels and Quarterman

The **icmpShowInit()** routine links the ICMP show facility into the VxWorks system. This is performed automatically if **INCLUDE_NET_SHOW** is defined in **configAll.h**.

SEE ALSO **netLib**, **netShow**, *Network Programmer's Guide*

ideDrv

NAME `ideDrv` – IDE disk device driver

ROUTINES `ideDrv()` – initialize the IDE driver
`ideDevCreate()` – create a device for a IDE disk
`ideRawio()` – provide raw I/O access

DESCRIPTION This is the driver for the IDE used on the PC 386/486.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. However, two routines must be called directly: `ideDrv()` to initialize the driver, and `ideDevCreate()` to create devices.

Before the driver can be used, it must be initialized by calling `ideDrv()`. This routine should be called exactly once, before any reads, writes, or calls to `ideDevCreate()`. Normally, it is called from `usrRoot()` in `usrConfig.c`.

The routine `ideRawio()` provides physical I/O access. Its first argument is a drive number, 0 or 1; the second argument is a pointer to an `IDE_RAW` structure.

NOTE Format is not supported, because IDE disks are already formatted, and bad sectors are mapped.

SEE ALSO *VxWorks Programmer's Guide: I/O System*

ifLib

NAME `ifLib` – network interface library

ROUTINES `ifAddrAdd()` – Add an interface address for a network interface
`ifAddrSet()` – set an interface address for a network interface
`ifAddrGet()` – get the Internet address of a network interface
`ifBroadcastSet()` – set the broadcast address for a network interface
`ifBroadcastGet()` – get the broadcast address for a network interface
`ifDstAddrSet()` – define an address for the other end of a point-to-point link
`ifDstAddrGet()` – get the Internet address of a point-to-point peer
`ifMaskSet()` – define a subnet for a network interface
`ifMaskGet()` – get the subnet mask for a network interface
`ifFlagChange()` – change the network interface flags

if_cpm

ifFlagSet() – specify the flags for a network interface
ifFlagGet() – get the network interface flags
ifMetricSet() – specify a network interface hop count
ifMetricGet() – get the metric for a network interface
ifRouteDelete() – delete routes associated with a network interface
ifunit() – map an interface name to an interface structure pointer

| | |
|----------------------|---|
| DESCRIPTION | This library contains routines to configure the network interface parameters. Generally, each routine corresponds to one of the functions of the UNIX command ifconfig . |
| INCLUDE FILES | ifLib.h |
| SEE ALSO | hostLib , <i>VxWorks Programmer's Guide: Network</i> |

if_cpm

| | |
|---------------------------|--|
| NAME | if_cpm – Motorola CPM core network interface driver |
| ROUTINES | <i>cpmattach()</i> – publish the cpm network interface and initialize the driver <i>cpmStartOutput()</i> – output packet to network interface device |
| DESCRIPTION | <p>This module implements the driver for the Motorola CPM core Ethernet network interface used in the M68EN360 and PPC800-series communications controllers.</p> <p>The driver is designed to support the Ethernet mode of an SCC residing on the CPM processor core. It is generic in the sense that it does not care which SCC is being used, and it supports up to four individual units per board.</p> <p>The driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below.</p> <p>This network interface driver does not include support for trailer protocols or data chaining. However, buffer loaning has been implemented in an effort to boost performance. This driver provides support for four individual device units.</p> <p>This driver maintains cache coherency by allocating buffer space using the <i>cacheDmaMalloc()</i> routine. It is assumed that cache-safe memory is returned; this driver does not perform cache flushing and invalidating.</p> |
| BOARD LAYOUT | This device is on-chip. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | This driver presents the standard WRS network driver API: the device unit must be |

attached and initialized with the *cpmattach()* routine.

The only user-callable routine is *cpmattach()*, which publishes the *cpm* interface and initializes the driver structures.

TARGET-SPECIFIC PARAMETERS

These parameters are passed to the driver via *cpmattach()*.

address of SCC parameter RAM

This parameter is the address of the parameter RAM used to control the SCC. Through this address, and the address of the SCC registers (see below), different network interface units are able to use different SCCs without conflict. This parameter points to the internal memory of the chip where the SCC physically resides, which may not necessarily be the master chip on the target board.

address of SCC registers

This parameter is the address of the registers used to control the SCC. Through this address, and the address of the SCC parameter RAM (see above), different network interface units are able to use different SCCs without conflict. This parameter points to the internal memory of the chip where the SCC physically resides, which may not necessarily be the master chip on the target board.

interrupt-vector offset

This driver configures the SCC to generate hardware interrupts for various events within the device. The interrupt-vector offset parameter is used to connect the driver's ISR to the interrupt through a call to *intConnect()*.

address of transmit and receive buffer descriptors

These parameters indicate the base locations of the transmit and receive buffer descriptor (BD) rings. Each BD takes up 8 bytes of dual-ported RAM, and it is the user's responsibility to ensure that all specified BDs will fit within dual-ported RAM. This includes any other BDs the target board may be using, including other SCCs, SMCs, and the SPI device. There is no default for these parameters; they must be provided by the user.

number of transmit and receive buffer descriptors

The number of transmit and receive buffer descriptors (BDs) used is configurable by the user upon attaching the driver. Each buffer descriptor resides in 8 bytes of the chip's dual-ported RAM space, and each one points to a 1520-byte buffer in regular RAM. There must be a minimum of two transmit and two receive BDs. There is no maximum number of buffers, but there is a limit to how much the driver speed increases as more buffers are added, and dual-ported RAM space is at a premium. If this parameter is "NULL", a default value of 32 BDs is used.

base address of buffer pool

This parameter is used to notify the driver that space for the transmit and receive buffers need not be allocated, but should be taken from a cache-coherent private memory space provided by the user at the given address. The user should be aware that memory used for buffers must be 4-byte aligned and non-cacheable. All the

buffers must fit in the given memory space; no checking is performed. This includes all transmit and receive buffers (see above) and an additional 16 receive loaner buffers. If the number of receive BDs is less than 16, that number of loaner buffers is used. Each buffer is 1520 bytes. If this parameter is "NONE," space for buffers is obtained by calling `cacheDmaMalloc()` in `cpmattach()`.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires seven external support functions:

STATUS sysCpmEnetEnable (int unit)

This routine is expected to perform any target-specific functions required to enable the Ethernet controller. These functions typically include enabling the Transmit Enable signal (TENA) and connecting the transmit and receive clocks to the SCC. The driver calls this routine, once per unit, from the `cpmInit()` routine.

void sysCpmEnetDisable (int unit)

This routine is expected to perform any target-specific functions required to disable the Ethernet controller. This usually involves disabling the Transmit Enable (TENA) signal. The driver calls this routine from the `cpmReset()` routine each time a unit is disabled.

STATUS sysCpmEnetCommand (int unit, UINT16 command)

This routine is expected to issue a command to the Ethernet interface controller. The driver calls this routine to perform basic commands, such as restarting the transmitter and stopping reception.

void sysCpmEnetIntEnable (int unit)

This routine is expected to enable the interrupt for the Ethernet interface specified by *unit*.

void sysCpmEnetIntDisable (int unit)

This routine is expected to disable the interrupt for the Ethernet interface specified by *unit*.

void sysCpmEnetIntClear (int unit)

This routine is expected to clear the interrupt for the Ethernet interface specified by *unit*.

STATUS sysCpmEnetAddrGet (int unit, UINT8 * addr)

The driver expects this routine to provide the 6-byte Ethernet hardware address that will be used by *unit*. This routine must copy the 6-byte address to the space provided by *addr*. This routine is expected to return OK on success, or ERROR. The driver calls this routine, once per unit, from the `cpmInit()` routine.

SYSTEM RESOURCE USAGE

This driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector

- 0 bytes in the initialized data section (data)
- 1272 bytes in the uninitialized data section (BSS)

The data and BSS sections are quoted for the CPU32 architecture and may vary for other architectures. The code size (text) varies greatly between architectures, and is therefore not quoted here.

If the driver allocates the memory shared with the Ethernet device unit, it does so by calling the *cacheDmaMalloc()* routine. For the default case of 32 transmit buffers, 32 receive buffers, and 16 loaner buffers, the total size requested is 121,600 bytes. If a non-cacheable memory region is provided by the user, the size of this region should be this amount, unless the user has specified a different number of transmit or receive BDs.

This driver can operate only if the shared memory region is non-cacheable, or if the hardware implements bus snooping. The driver cannot maintain cache coherency for the device because the buffers are asynchronously modified by both the driver and the device, and these fields may share the same cache line. Additionally, the chip's dual ported RAM must be declared as non-cacheable memory where applicable.

SEE ALSO

ifLib, *Motorola MC68EN360 User's Manual*, *Motorola MPC860 User's Manual*, *Motorola MPC821 User's Manual*

if_cs

NAME

if_cs – Crystal Semiconductor CS8900 network interface driver

ROUTINES

csAttach() – publish the **cs** network interface and initialize the driver.
csShow() – shows statistics for the **cs** network interface

DESCRIPTION

This module implements a driver for a Crystal Semiconductor CS8900 Ethernet controller chip.

The CS8900 is a single chip Ethernet controller with a direct ISA bus interface which can operate in either memory space or I/O space. It also supports a direct interface to a host DMA controller to transfer receive frames to host memory. The device has a 4K RAM which is used for transmit, and receive buffers; a serial EEPROM interface; and both 10BASE-T/AUI port support.

This driver is capable of supporting both memory mode and I/O mode operations of the chip. When configured for memory mode, the internal RAM of the chip is mapped to a contiguous 4K address block, providing the CPU direct access to the internal registers and frame buffers. When configured for I/O mode, the internal registers are accessible through eight contiguous, 16-bit I/O ports. The driver also supports an interface to an EEPROM containing device configuration.

While the DMA slave mode is supported by the device for receive frame transfers, this driver does not enable DMA.

This network interface driver does not support output hook routines, because to do so requires that an image of the transmit packet be built in memory before the image is copied to the CS8900 chip. It is much more efficient to copy the image directly from the mbuf chain to the CS8900 chip. However, this network interface driver does support input hook routines.

CONFIGURATION The defined I/O address and IRQ in `config.h` must match the one stored in EEPROM by the vendor's DOS utility program.

The I/O Address parameter is the only required `csAttach()` parameter. If the CS8900 chip has a EEPROM attached, then the I/O Address parameter, passed to the `csAttach()` routine, must match the I/O address programmed into the EEPROM. If the CS8900 chip does not have a EEPROM attached, then the I/O Address parameter must be 0x300.

The Interrupt Level parameter must have one of the following values:

- 0 - Get interrupt level from EEPROM
- 5 - IRQ 5
- 10 - IRQ 10
- 11 - IRQ 11
- 12 - IRQ 12

If the Interrupt Vector parameter is zero, then the network interface driver derives the interrupt vector from the interrupt level if possible. It is possible to derive the interrupt vector in an IBM PC compatible system. This parameter is present for systems which are not IBM PC compatible.

The Memory Address parameter specifies the base address of the CS8900 chip's memory buffer (PacketPage). If the Memory Address parameter is not zero, then the CS8900 chip operates in memory mode at the specified address. If the Memory Address parameter is zero, then the CS8900 chip operates in the mode specified by the EEPROM or the Configuration Flags parameter.

The Media Type parameter must have one of the following values:

- 0 - Get media type from EEPROM
- 1 - AUI (Thick Cable)
- 2 - BNC 10Base2 (Thin Cable)
- 3 - RJ45 10BaseT (Twisted Pair)

The Configuration Flags parameter is usually passed to the `csAttach()` routine as zero and the Configuration Flags information is retrieved from the EEPROM. The bits in the Configuration Flags parameter are usually specified by a hardware engineer and not by the end user. However, if the CS8900 chip does not have a EEPROM attached, then this information must be passed as a parameter to the `csAttach()` routine. The Configuration Flags are:

- 0x8000 - CS_CFGFLG_NOT_EEPROM Don't get Config. Flags from the EEPROM

| | |
|-----------------------------|--|
| 0x0001 - CS_CFGFLG_MEM_MODE | Use memory mode to access the chip |
| 0x0002 - CS_CFGFLG_USE_SA | Use system addr to qualify MEMCS16 signal |
| 0x0004 - CS_CFGFLG_IOCHRDY | Use IO Channel Ready signal to slow access |
| 0x0008 - CS_CFGFLG_DCDC_POL | The DC/DC conv. enable pin is active high |
| 0x0010 - CS_CFGFLG_FDX | 10BaseT is full duplex |

If configuration flag information is passed to the *csAttach()* routine, then the CS_CFGFLG_NOT_EEPROM flag should be set. This ensures that the Configuration Flags parameter is not zero, even if all specified flags are zero.

If the Memory Address parameter is not zero and the Configuration Flags parameter is zero, then the CS8900 network interface driver implicitly sets the CS_CFGFLG_MEM_MODE flag and the CS8900 chip operates in memory mode. However, if the Configuration Flags parameter is not zero, then the CS8900 chip operates in memory mode only if the CS_CFGFLG_MEM_MODE flag is explicitly set. If the Configuration Flags parameter is not zero and the CS_CFGFLG_MEM_MODE flag is not set, then the CS8900 chip operates in I/O mode.

The Ethernet Address parameter is usually passed to the *csAttach()* routine as zero and the Ethernet address is retrieved from the EEPROM. The Ethernet address (also called hardware address and individual address) is usually supplied by the adapter manufacturer and is stored in the EEPROM. However, if the CS8900 chip does not have a EEPROM attached, then the Ethernet address must be passed as a parameter to the *csAttach()* routine. The Ethernet Address parameter, passed to the *csAttach()* routine, contains the address of a NULL terminated string. The string consists of 6 hexadecimal numbers separated by colon characters. Each hexadecimal number is in the range 00 – FF. An example of this string is:

```
"00:24:20:10:FF:2A"
```

BOARD LAYOUT This device is soft-configured. No jumpering diagram is required.

EXTERNAL INTERFACE

The only user-callable routines are *csAttach()*:

csAttach()

publishes the *cs* interface and initializes the driver and device.

The network interface driver includes a show routine, called *csShow()*, which displays driver configuration and statistics information. To invoke the show routine, type at the shell prompt:

```
-> csShow
```

To reset the statistics to zero, type at the shell prompt:

```
-> csShow 0, 1
```

Another routine that you may find useful is:

```
-> ifShow "cs0"
```

EXTERNAL ROUTINES

For debugging purposes, this driver calls *logMsg()* to print error and debugging information. This will cause the **logLib** library to be linked with any image containing this driver.

This driver needs the following macros defined for proper execution. Each has a default definition that assumes a PC386/PC486 system and BSP.

The macro **CS_IN_BYTE**(reg,pAddr) reads one byte from the I/O address **reg**, placing the result at address **pAddr**. There is no status result from this operation, we assume the operation completes normally, or a bus exception will occur. By default, this macro assumes there is a BSP routine *sysInByte()* to perform the I/O operation.

The macro **CS_IN_WORD**(reg,pAddr) read a short word (2 bytes) from the I/O address **reg**, storing the result at address **pAddr**. We assume this completes normally, or causes a bus exception. The default declaration assumes a BSP routine *sysInWord()* to perform the operation.

The macro **CS_OUT_WORD**(reg,data) writes a short word value **data** at the I/O address **reg**. The default declaration assumes a BSP routine *sysOutWord()*.

The macro **CS_INT_ENABLE**(level, pResult) is used to enable the interrupt level passed as an argument to csAttach. The default definition call the BSP routine *sysIntEnablePIC(level)*. The STATUS return value from the actual routine is stored at **pResult** for the driver to examine.

The macro **CS_INT_CONNECT**(ivec,rtn,arg,pResult) macro is used to connect the driver interrupt routine to the vector provided as an argument to csAttach (after translation by **INUM_TO_IVEC**). The default definition calls the cpu architecture routine *intConnect()*.

The macro **CS_IRQ0_VECTOR**(pAddr) is used to fetch the base vector for the interrupt level mechanism. If the int vector argument to csAttach is zero, then the driver will compute a vector number by adding the interrupt level to the value returned by this macro. If the user supplies a non-zero interrupt vector number, then this macro is not used. The default definition of this macro fetches the base vector number from a global value called **sysVectorIRQ0**.

The macro **CS_MSEC_DELAY**(msec) is used to delay execution for a specified number of milliseconds. The default definition uses taskDelay to suspend task for some number of clock ticks. The resolution of the system clock is usually around 16 milliseconds (msecs), which is fairly coarse.

if_dc

| | |
|---------------------------|---|
| NAME | if_dc – DEC 21x4x Ethernet LAN network interface driver |
| ROUTINES | <i>dcattach()</i> – publish the dc network interface. <i>dcReadAllRom()</i> – read entire serial rom <i>dcViewRom()</i> – display lines of serial ROM for dec21140 <i>dcCsrShow()</i> – display dec 21040/21140 status registers 0 thru 15 |
| DESCRIPTION | <p>This module implements an ethernet interface driver for the DEC 21x4x family, and currently supports the following variants -- 21040, 21140, and 21140A.</p> <p>The DEC 21x4x PCI Ethernet controllers are inherently little-endian since they are designed for a little-endian PCI bus. While the 21040 only supports a 10Mbps interface, other members of this family are dual-speed devices which support both 10 and 100 Mbps.</p> <p>This driver is designed to be moderately generic, operating unmodified across the range of architectures and targets supported by VxWorks; and on multiple versions of the dec21x4x family. To achieve this, the driver takes several parameters, and external support routines which are detailed below. Also stated below are assumptions made by the driver of the hardware, and if any of these assumptions are not true for your hardware, the driver will probably not function correctly.</p> <p>This driver supports up to 4 ethernet units per CPU, and can be configured for either big-endian or little-endian architectures. It contains error-recovery code to handle known device errata related to DMA activity.</p> <p>On a dec21040, this driver configures the 10BASE-T interface by default and waits for two seconds to check the status of the link. If the link status is "fail," it then configures the AUI interface.</p> <p>The dec21140, and dec21140A devices support both 10 and 100Mbps and also a variety of MII and non-MII PHY interfaces. This driver reads a DEC version 2.0 SROM device for PHY initialization information, and automatically configures an appropriate active PHY media.</p> |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | <p>This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine; there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the initialization routine is NULL.</p> <p>The only user-callable routine is <i>dcattach()</i>, which publishes the dc interface and initializes the driver and device.</p> |

TARGET-SPECIFIC PARAMETERS

bus mode

This parameter is a global variable that can be modified at run-time.

The LAN control register #0 determines the bus mode of the device, allowing the support of big-endian and little-endian architectures. This parameter, defined as "ULONG dcCSR0Bmr", is the value that will be placed into device control register #0. The default is mode is little endian. For information about changing this parameter, see the manual *DEC Local Area Network Controller DEC21040 or DEC21140 for PCI*.

base address of device registers

This parameter is passed to the driver by *dcattach()*.

interrupt vector

This parameter is passed to the driver by *dcattach()*.

This driver configures the device to generate hardware interrupts for various events within the device; thus it contains an interrupt handler routine. The driver calls *intConnect()* to connect its interrupt handler to the interrupt vector generated as a result of the device interrupt.

interrupt level

This parameter is passed to the driver by *dcattach()*.

Some targets use additional interrupt controller devices to help organize and service the various interrupt sources. This driver avoids all board-specific knowledge of such devices. During the driver's initialization, the external routine *sysLanIntEnable()* is called to perform any board-specific operations required to allow the servicing of a device interrupt. For a description of *sysLanIntEnable()*, see "External Support Requirements" below.

This parameter is passed to the external routine.

shared memory address

This parameter is passed to the driver by *dcattach()*.

The DEC 21x4x device is a DMA type of device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the DEC 21x4x. It assumes that this shared memory is directly available to it without any arbitration or timing concerns.

This parameter can be used to specify an explicit memory region for use by the DEC 21x4x device. This should be done on hardware that restricts the DEC 21x4x device to a particular memory region. The constant NONE can be used to indicate that there are no memory limitations, in which case, the driver attempts to allocate the shared memory from the system space.

shared memory size

This parameter is passed to the driver by *dcattach()*.

This parameter can be used to explicitly limit the amount of shared memory (bytes)

this driver will use. The constant NONE can be used to indicate no specific size limitation. This parameter is used only if a specific memory region is provided to the driver.

shared memory width

This parameter is passed to the driver by *dcattach()*.

Some target hardware that restricts the shared memory region to a specific location also restricts the access width to this region by the CPU. On these targets, performing an access of an invalid width will cause a bus error.

This parameter can be used to specify the number of bytes of access width to be used by the driver during access to the shared memory. The constant NONE can be used to indicate no restrictions.

Current internal support for this mechanism is not robust; implementation may not work on all targets requiring these restrictions.

shared memory buffer size

This parameter is passed to the driver by *dcattach()*.

The driver and DEC 21x4x device exchange network data in buffers. This parameter permits the size of these individual buffers to be limited. A value of zero indicates that the default buffer size should be used. The default buffer size is large enough to hold a maximum-size Ethernet packet.

pci Memory base

This parameter is passed to the driver by *dcattach()*. This parameter gives the base address of the main memory on the PCI bus.

dcOpMode

This parameter is passed to the driver by *dcattach()*. This parameter gives the mode of initialization of the device. The mode flags for both the DEC21040 and DEC21140 interfaces are listed below.

```
DC_PROMISCUOUS_FLAG 0x01
DC_MULTICAST_FLAG   0x02
```

The mode flags specific to the DEC21140 interface are listed below.

```
DC_100_MB_FLAG      0x04
DC_21140_FLAG        0x08
DC_SCRAMBLER_FLAG   0x10
DC_PCS_FLAG          0x20
DC_PS_FLAG           0x40
DC_FULLDUPLEX_FLAG  0x10
```

Loopback mode flags:

```
DC_ILOOPB_FLAG      0x100
DC_ELOOPB_FLAG      0x200
DC_HBE_FLAG          0x400
```

if_eex

Ethernet address

This is obtained by the driver by reading an ethernet ROM register or the DEC serial ROM.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires one external support function:

```
void sysLanIntEnable (int level)
```

This routine provides a target-specific enable of the interrupt for the DEC 21x4x device. Typically, this involves interrupt controller hardware, either internal or external to the CPU.

This routine is called once via the macro **SYS_INT_ENABLE**.

SEE ALSO

ifLib, *DECchip 21040 or 21140 Ethernet LAN Controller for PCI*.

if_eex

NAME

if_eex – Intel EtherExpress 16 network interface driver

ROUTINES

eexattach() – publish the **eex** network interface and initialize the driver and device
eexTxStartup() – start output on the chip

DESCRIPTION

This module implements the Intel EtherExpress 16 PC network interface card driver. It is specific to that board as used in PC 386/486 hosts. This driver is written using the device's I/O registers exclusively.

SIMPLIFYING ASSUMPTIONS

This module assumes a little-endian host (80x86); thus, no endian adjustments are needed to manipulate the 82586 data structures (little-endian).

The on-board memory is assumed to be sufficient; thus, no provision is made for additional buffering in system memory.

The "frame descriptor" and "buffer descriptor" structures can be bound into permanent pairs by pointing each FD at a "chain" of one BD of MTU size. The 82586 receive algorithm fills exactly one BD for each FD; it looks to the NEXT FD in line for the next BD.

The transmit and receive descriptor lists are permanently linked into circular queues partitioned into sublists designated by the **EEX_LIST** headers in the driver control structure. Empty partitions have NULL pointer fields. EL bits are set as needed to tell the 82586 where a partition ends. The lists are managed in strict FIFO fashion; thus the link fields are never modified, just ignored if a descriptor is at the end of a list partition.

BOARD LAYOUT This device is soft-configured. No jumpering diagram is required.

EXTERNAL INTERFACE

This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine and there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the *init()* routine is NULL.

There is one user-callable routine, *eexattach()*. For details on usage, see the manual entry for this routine.

EXTERNAL SUPPORT REQUIREMENTS

None.

SYSTEM RESOURCE USAGE

- one mutual exclusion semaphore
- one interrupt vector
- one watchdog timer.
- 8 bytes in the initialized data section (data)
- 912 bytes in the uninitialized data section (bss)

The data and bss sections are quoted for the MC68020 architecture and may vary for other architectures. The code size (text) will vary widely between architectures, and is thus not quoted here.

The device contains on-board buffer memory; no system memory is required for buffering.

TUNING HINTS

The only adjustable parameter is the number of TFDs to create in adapter buffer memory. The total number of TFDs and RFDs is 21, given full-frame buffering and the sizes of the auxiliary structures. *eexattach()* requires at least `MIN_NUM_RFDS` RFDs to exist. More than ten TFDs is not sensible in typical circumstances.

SEE ALSO

ifLib

if_ei

NAME

if_ei – Intel 82596 Ethernet network interface driver

ROUTINES

eiattach() – publish the *ei* network interface and initialize the driver and device
eiTxStartup() – start output on the chip

DESCRIPTION

This module implements the Intel 82596 Ethernet network interface driver.

if_ei

This driver is designed to be moderately generic, operating unmodified across the range of architectures and targets supported by VxWorks. To achieve this, this driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below.

This driver can run with the device configured in either big-endian or little-endian modes. Error recovery code has been added to deal with some of the known errata in the A0 version of the device. This driver supports up to four individual units per CPU.

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine; there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the initialization routine is NULL.

The only user-callable routine is *eiattach()*, which publishes the *eiinterface* and initializes the driver and device.

TARGET-SPECIFIC PARAMETERS

the *sysbus* value

This parameter is passed to the driver by *eiattach()*. The Intel 82596 requires this parameter during initialization. This parameter tells the device about the system bus, hence the name "sysbus." To determine the correct value for a target, refer to the document *Intel 32-bit Local Area Network (LAN) Component User's Manual*.

interrupt vector

This parameter is passed to the driver by *eiattach()*. The Intel 82596 generates hardware interrupts for various events within the device; thus it contains an interrupt handler routine. This driver calls *intConnect()* to connect its interrupt handler to the interrupt vector generated as a result of the 82596 interrupt.

shared memory address

This parameter is passed to the driver by *eiattach()*. The Intel 82596 device is a DMA type device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the 82596.

This parameter can be used to specify an explicit memory region for use by the 82596. This should be done on targets that restrict the 82596 to a particular memory region. The constant NONE can be used to indicate that there are no memory limitations, in which case, the driver attempts to allocate the shared memory from the system space.

number of Receive and Transmit Frame Descriptors

These parameters are passed to the driver by *eiattach()*. The Intel 82596 accesses frame descriptors in memory for each frame transmitted or received. The number of

frame descriptors at run-time can be configured using these parameters.

Ethernet address

This parameter is obtained by a call to an external support routine. During initialization, the driver needs to know the Ethernet address for the Intel 82596 device. The driver calls the external support routine, *sysEnetAddrGet()*, to obtain the Ethernet address. For a description of *sysEnetAddrGet()*, see "External Support Requirements" below.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires seven external support functions:

STATUS *sysEnetAddrGet* (int unit, char *pCopy)

This routine provides the six-byte Ethernet address used by *unit*. It must copy the six-byte address to the space provided by *pCopy*. This routine returns OK, or ERROR if it fails. The driver calls this routine, once per unit, using *eiattach()*.

STATUS *sys596Init* (int unit)

This routine performs any target-specific initialization required before the 82596 is initialized. Typically, it is empty. This routine must return OK, or ERROR if it fails. The driver calls this routine, once per unit, using *eiattach()*.

void *sys596Port* (int unit, int cmd, UINT32 addr)

This routine provides access to the special port function of the 82596. It delivers the command and address arguments to the port of the specified unit. The driver calls this routine primarily during initialization, but may also call it during error recovery procedures.

void *sys596ChanAtn* (int unit)

This routine provides the channel attention signal to the 82596, for the specified *unit*. The driver calls this routine frequently throughout all phases of operation.

void *sys596IntEnable* (int unit), **void** *sys596IntDisable* (int unit)

These routines enable or disable the interrupt from the 82596 for the specified *unit*. Typically, this involves interrupt controller hardware, either internal or external to the CPU. Since the 82596 itself has no mechanism for controlling its interrupt activity, these routines are vital to the correct operation of the driver. The driver calls these routines throughout normal operation to protect certain critical sections of code from interrupt handler intervention.

void *sys596IntAck* (int unit)

This routine must perform any required interrupt acknowledgment or clearing. Typically, this involves an operation to some interrupt control hardware. Note that the INT signal from the 82596 behaves in an "edge-triggered" mode; therefore, this routine typically clears a latch within the control circuitry. The driver calls this routine from the interrupt handler.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

if_eihk

- one mutual exclusion semaphore
- one interrupt vector
- one watchdog timer.
- 8 bytes in the initialized data section (data)
- 912 bytes in the uninitialized data section (BSS)

The above data and BSS requirements are for the MC68020 architecture and may vary for other architectures. Code size (text) varies greatly between architectures and is therefore not quoted here.

The driver uses *cacheDmaMalloc()* to allocate memory to share with the 82596. The fixed-size pieces in this area total 160 bytes. The variable-size pieces in this area are affected by the configuration parameters specified in the *eiattach()* call. The size of one RFD (Receive Frame Descriptor) is 1536 bytes. The size of one TFD (Transmit Frame Descriptor) is 1534 bytes. For more information about RFDs and TFDs, see the *Intel 82596 User's Manual*.

The 82596 can be operated only if this shared memory region is non-cacheable or if the hardware implements bus snooping. The driver cannot maintain cache coherency for the device because fields within the command structures are asynchronously modified by both the driver and the device, and these fields may share the same cache line.

TUNING HINTS

The only adjustable parameters are the number of TFDs and RFDs that will be created at run-time. These parameters are given to the driver when *eiattach()* is called. There is one TFD and one RFD associated with each transmitted frame and each received frame respectively. For memory-limited applications, decreasing the number of TFDs and RFDs may be desirable. Increasing the number of TFDs will provide no performance benefit after a certain point. Increasing the number of RFDs will provide more buffering before packets are dropped. This can be useful if there are tasks running at a higher priority than the net task.

CAVEAT

This driver does not support promiscuous mode.

SEE ALSO

ifLib, *Intel 82596 User's Manual*, *Intel 32-bit Local Area Network (LAN) Component User's Manual*

if_eihk

NAME

if_eihk – Intel 82596 Ethernet network interface driver for hkv3500

ROUTINES

eihkattach() – publish the **ei** network interface and initialize the driver and device
eiTxStartup() – start output on the chip
eiInt() – entry point for handling interrupts from the 82596

DESCRIPTION This module implements a hkv3500 specific Intel 82596 Ethernet network interface driver. This driver is derived from the generic *if_ei* ethernet driver to support hkv3500 target board. The receive buffer scheme has been modified from a simplified memory structure to a flexible memory structure so that receive buffers can be word-aligned, and thus support buffer loaning on a MIPS CPU architecture.

The driver requires several target-specific parameters, and some external support routines which are detailed below.

This driver can run with the device configured in either big-endian or little-endian modes. Error recovery code has been added to deal with some of the known errata in the A0 version of the device. This driver supports up to four individual units per CPU.

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine; there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the initialization routine is NULL.

The only user-callable routine is *eihkattach()*, which publishes the *ei* interface and initializes the driver and device.

TARGET-SPECIFIC PARAMETERS

the *sysbus* value

This parameter is passed to the driver by *eihkattach()*.

The Intel 82596 requires this parameter during initialization. This parameter tells the device about the system bus, hence the name "sysbus." To determine the correct value for a target, refer to the document *Intel 32-bit Local Area Network (LAN) Component User's Manual*.

interrupt vector

This parameter is passed to the driver by *eihkattach()*.

The Intel 82596 generates hardware interrupts for various events within the device; thus it contains an interrupt handler routine. This driver calls *intConnect()* to connect its interrupt handler to the interrupt vector generated as a result of the 82596 interrupt.

shared memory address

This parameter is passed to the driver by *eihkattach()*.

The Intel 82596 device is a DMA type device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the 82596.

This parameter can be used to specify an explicit memory region for use by the 82596.

This should be done on targets that restrict the 82596 to a particular memory region. The constant NONE can be used to indicate that there are no memory limitations, in which case, the driver attempts to allocate the shared memory from the system space.

number of Receive and Transmit Frame Descriptors

These parameters are passed to the driver by *eihkattach()*.

The Intel 82596 accesses frame descriptors in memory for each frame transmitted or received. The number of frame descriptors at run-time can be configured using these parameters.

Ethernet address

This parameter is obtained by a call to an external support routine.

During initialization, the driver needs to know the Ethernet address for the Intel 82596 device. The driver calls the external support routine, *sysEnetAddrGet()*, to obtain the Ethernet address. For a description of *sysEnetAddrGet()*, see "External Support Requirements" below.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires seven external support functions:

STATUS sysEnetAddrGet (int unit, char *pCopy)

This routine provides the six-byte Ethernet address used by *unit*. It must copy the six-byte address to the space provided by *pCopy*. This routine returns OK, or ERROR if it fails. The driver calls this routine, once per unit, using *eihkattach()*.

STATUS sys596Init (int unit, SCB *pScb)

This routine performs any target-specific initialization required before the 82596 is initialized. Typically, it is empty. This routine must return OK, or ERROR if it fails. The driver calls this routine, once per unit, using *eihkattach()*.

void sys596Port (int unit, int cmd, UINT32 addr)

This routine provides access to the special port function of the 82596. It delivers the command and address arguments to the port of the specified unit. The driver calls this routine primarily during initialization, but may also call it during error recovery procedures.

void sys596ChanAtn (int unit)

This routine provides the channel attention signal to the 82596, for the specified *unit*. The driver calls this routine frequently throughout all phases of operation.

void sys596IntEnable (int unit), void sys596IntDisable (int unit)

These routines enable or disable the interrupt from the 82596 for the specified *unit*. Typically, this involves interrupt controller hardware, either internal or external to the CPU. Since the 82596 itself has no mechanism for controlling its interrupt activity, these routines are vital to the correct operation of the driver. The driver calls these routines throughout normal operation to protect certain critical sections of code from interrupt handler intervention.

```
void sys596IntAck (int unit)
```

This routine must perform any required interrupt acknowledgment or clearing. Typically, this involves an operation to some interrupt control hardware. Note that the INT signal from the 82596 behaves in an "edge-triggered" mode; therefore, this routine typically clears a latch within the control circuitry. The driver calls this routine from the interrupt handler.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- one watchdog timer.
- 8 bytes in the initialized data section (data)
- 912 bytes in the uninitialized data section (BSS)

The above data and BSS requirements are for the MC68020 architecture and may vary for other architectures. Code size (text) varies greatly between architectures and is therefore not quoted here.

The driver uses *cacheDmaMalloc()* to allocate memory to share with the 82596. The fixed-size pieces in this area total 160 bytes. The variable-size pieces in this area are affected by the configuration parameters specified in the *eihkattach()* call. The size of one RFD (Receive Frame Descriptor) is 1536 bytes. The size of one TFD (Transmit Frame Descriptor) is 1534 bytes. For more information about RFDs and TFDs, see the *Intel 82596 User's Manual*.

The 82596 can be operated only if this shared memory region is non-cacheable or if the hardware implements bus snooping. The driver cannot maintain cache coherency for the device because fields within the command structures are asynchronously modified by both the driver and the device, and these fields may share the same cache line.

TUNING HINTS

The only adjustable parameters are the number of TFDs and RFDs that will be created at run-time. These parameters are given to the driver when *eihkattach()* is called. There is one TFD and one RFD associated with each transmitted frame and each received frame respectively. For memory-limited applications, decreasing the number of TFDs and RFDs may be desirable. Increasing the number of TFDs will provide no performance benefit after a certain point. Increasing the number of RFDs will provide more buffering before packets are dropped. This can be useful if there are tasks running at a higher priority than the net task.

SEE ALSO

ifLib, *Intel 82596 User's Manual*, *Intel 32-bit Local Area Network (LAN) Component User's Manual*

if_elc

| | |
|---------------------------|---|
| NAME | if_elc – SMC 8013WC Ethernet network interface driver |
| ROUTINES | <i>elcattach()</i> – publish the elc network interface and initialize the driver and device <i>elcPut()</i> – copy a packet to the interface. <i>elcShow()</i> – display statistics for the SMC 8013WC elc network interface |
| DESCRIPTION | This module implements the SMC 8013WC network interface driver. |
| BOARD LAYOUT | The W1 jumper should be set in position SOFT. The W2 jumper should be set in position NONE/SOFT. |
| CONFIGURATION | The I/O address, RAM address, RAM size, and IRQ levels are defined in config.h . The I/O address must match the one stored in EEROM. The configuration software supplied by the manufacturer should be used to set the I/O address. IRQ levels 2,3,4,5,7,9,10,11,15 are supported. Thick Ethernet (AUI) and Thin Ethernet (BNC) are configurable by changing the macro CONFIG_ELC in config.h . |
| EXTERNAL INTERFACE | The only user-callable routines are <i>elcattach()</i> and <i>elcShow()</i> : <i>elcattach()</i> publishes the elc interface and initializes the driver and device. <i>elcShow()</i> displays statistics that are collected in the interrupt handler. |
| SEE ALSO | if_elc |

if_elt

| | |
|--------------------|---|
| NAME | if_elt – 3Com 3C509 Ethernet network interface driver |
| ROUTINES | <i>eltattach()</i> – publish the elt interface and initialize the driver and device <i>eltTxOutputStart()</i> – start output on the board <i>eltShow()</i> – display statistics for the 3C509 elt network interface |
| DESCRIPTION | This module implements the 3Com 3C509 network adapter driver. |

The 3C509 (EtherLink®; III) is not well-suited for use in real-time systems. Its meager on-board buffering (4K total; 2K transmit, 2K receive) forces the host processor to service the board at a high priority. 3Com makes a virtue of this necessity by adding fancy lookahead support and adding the label "Parallel Tasking" to the outside of the box. Using 3Com's drivers, this board will look good in benchmarks that measure raw link speed. The board is greatly simplified by using the host CPU as a DMA controller.

BOARD LAYOUT This device is soft-configured by a DOS-hosted program supplied by the manufacturer. No jumpering diagram is required.

EXTERNAL INTERFACE

This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine and there is no separate initialization routine. Thus, in the global interface structure, the function pointer to the initialization routine is NULL.

There are two user-callable routines:

eltattach()

publishes the **elt** interface and initializes the driver and device.

eltShow()

displays statistics that are collected in the interrupt handler.

See the manual entries for these routines for more detail.

SYSTEM RESOURCE USAGE

- one mutual exclusion semaphore
- one interrupt vector
- 16 bytes in the uninitialized data section (bss)
- 180 bytes (plus overhead) of malloc'ed memory per unit
- 1530 bytes (plus overhead) of malloc'ed memory per frame buffer, minimum 5 frame buffers.

SHORTCUTS

The EISA and MCA versions of the board are not supported.

Attachment selection assumes the board is in power-on reset state; a warm restart will not clear the old attachment selection out of the hardware, and certain new selections may not clear it either. For example, if RJ45 was selected, the system is warm-booted, and AUI is selected, the RJ45 connector is still functional.

Attachment type selection is not validated against the board's capabilities, even though there is a register that describes which connectors exist.

The loaned buffer cluster type is **MC_EI**; no new type is defined yet.

Although it seems possible to put the transmitter into a non-functioning state, it is not obvious either how to do this or how to detect the resulting state. There is therefore no transmit watchdog timer.

No use is made of the tuning features of the board; it is possible that proper dynamic tuning would reduce or eliminate the receive overruns that occur when receiving under task control (instead of in the ISR).

TUNING HINTS More receive buffers (than the default 20) could help by allowing more loaning in cases of massive reception; four per receiving TCP connection plus four extras should be considered a minimum.

SEE ALSO **ifLib**

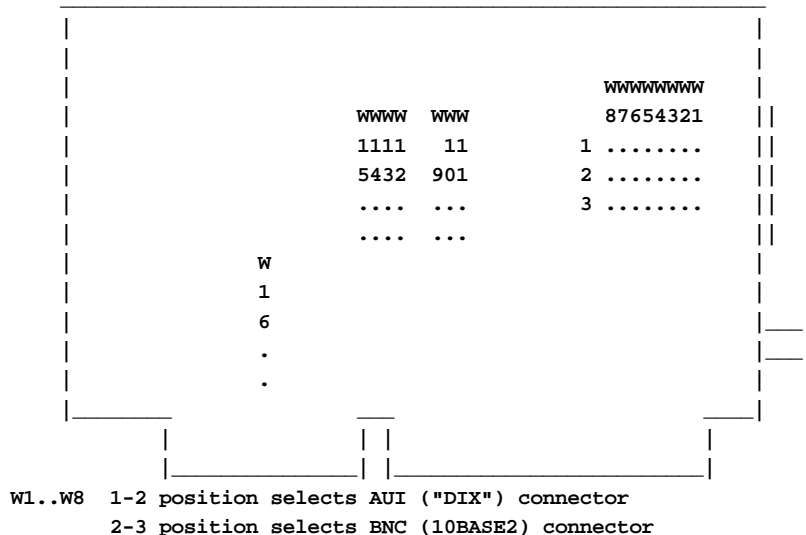
if_ene

NAME **if_ene** – Novell/Eagle NE2000 network interface driver

ROUTINES *eneattach()* – publish the **ene** network interface and initialize the driver and device
enePut() – copy a packet to the interface.
eneShow() – display statistics for the NE2000 **ene** network interface

DESCRIPTION This module implements the Novell/Eagle NE2000 network interface driver. There is one user-callable routine, *eneattach()*.

BOARD LAYOUT The diagram below shows the relevant jumpers for VxWorks configuration. Other compatible boards will be jumpered differently; many are jumperless.



```

W9..W11 YYN I/O address 300h, no boot ROM
        NYN I/O address 320h, no boot ROM
        YNN I/O address 340h, no boot ROM
        NNN I/O address 360h, no boot ROM
        YYY I/O address 300h, boot ROM at paragraph 0c800h
        NYY I/O address 320h, boot ROM at paragraph 0cc00h
        YNY I/O address 340h, boot ROM at paragraph 0d000h
        NNY I/O address 360h, boot ROM at ??? (invalid configuration?)
W12     Y   IRQ 2 (or 9 if you prefer)
W13     Y   IRQ 3
W14     Y   IRQ 4
W15     Y   IRQ 5 (note that only one of W12..W15 may be installed)
W16     Y   normal ISA bus timing
        N   timing for COMPAQ 286 portable, PS/2 Model 30-286, C&T
chipset

```

EXTERNAL INTERFACE

There are two user-callable routines:

eneattach()

publishes the **ene** interface and initializes the driver and device.

eneShow()

displays statistics that are collected in the interrupt handler.

See the manual entries for these routines for more detail.

SYSTEM RESOURCE USAGE

- one interrupt vector
- 16 bytes in the uninitialized data section (bss)
- 1752 bytes (plus overhead) of malloc'ed memory per unit attached

CAVEAT

This driver does not enable the twisted-pair connector on the Taiwanese ETHER-16 compatible board.

if_esmc

NAME

if_esmc - Ampro Ethernet2 SMC-91c9x Ethernet network interface driver

ROUTINES

esmccattach() - publish the **esmc** network interface and initialize the driver.

esmcpPut() - copy a packet to the interface.

esmcsShow() - display statistics for the **esmc** network interface

if_fei

NAME *if_fei* – Intel 82557 Ethernet network interface driver

ROUTINES *feiattach()* – publish the *fei* network interface

DESCRIPTION This module implements the Intel 82557 Ethernet network interface driver. This driver is designed to be moderately generic, operating unmodified across the entire range of architectures and targets supported by VxWorks. This driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below.

This driver supports up to four individual units.

EXTERNAL INTERFACE

The user-callable routine is *feiattach()*, which publishes the *fei* interface and performs some initialization.

After calling *feiattach()* to publish the interface, an initialization routine must be called to bring the device up to an operational state. The initialization routine is not a user-callable routine; upper layers call it when the interface flag is set to **UP**, or when the interface's IP address is set.

There is a global variable **feiIntConnect** which specifies the interrupt connect routine to be used depending on the BSP. This is by default set to *intConnect()* and the user can override this to use any other interrupt connect routine (say *pciIntConnect()*) in *sysHwInit()* or any device specific initialization routine called in *sysHwInit()*.

TARGET-SPECIFIC PARAMETERS

shared memory address

This parameter is passed to the driver via *feiattach()*.

The Intel 82557 device is a DMA-type device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the 82557.

This parameter can be used to specify an explicit memory region for use by the 82557. This should be done on targets that restrict the 82557 to a particular memory region. The constant **NONE** can be used to indicate that there are no memory limitations, in which case the driver attempts to allocate the shared memory from the system space.

number of Command, Receive, and Loanable-Receive Frame Descriptors

These parameters are passed to the driver via *feiattach()*.

The Intel 82557 accesses frame descriptors (and their associated buffers) in memory

for each frame transmitted or received. The number of frame descriptors can be configured at run-time using these parameters.

Ethernet address

This parameter is obtained by a call to an external support routine.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires the following external support function:

```
STATUS sys557Init (int unit, BOARD_INFO *pBoard)
```

This routine performs any target-specific initialization required before the 82557 device is initialized by the driver. The driver calls this routine every time it wants to [re]initialize the device. This routine returns OK, or ERROR if it fails.

SYSTEM RESOURCE USAGE

The driver uses *cacheDmaMalloc()* to allocate memory to share with the 82557. The size of this area is affected by the configuration parameters specified in the *feiattach()* call. The size of one RFD (Receive Frame Descriptor) is the same as one CFD (Command Frame Descriptor): 1536 bytes. For more information about RFDs and CFDs, see the *Intel 82557 User's Manual*.

Either the shared memory region must be non-cacheable, or else the hardware must implement bus snooping. The driver cannot maintain cache coherency for the device because fields within the command structures are asynchronously modified by both the driver and the device, and these fields may share the same cache line.

Additionally, this version of the driver does not handle virtual-to-physical or physical-to-virtual memory mapping.

TUNING HINTS

The only adjustable parameters are the number of Frame Descriptors that will be created at run-time. These parameters are given to the driver when *feiattach()* is called. There is one CFD and one RFD associated with each transmitted frame and each received frame, respectively. For memory-limited applications, decreasing the number of CFDs and RFDs may be desirable. Increasing the number of CFDs will provide no performance benefit after a certain point. Increasing the number of RFDs will provide more buffering before packets are dropped. This can be useful if there are tasks running at a higher priority than the net task.

SEE ALSO

ifLib, *Intel 82557 User's Manual*

if_fn

- NAME** `if_fn` – Fujitsu MB86960 NICE Ethernet network interface driver
- ROUTINES** `fnattach()` – publish the `fn` network interface and initialize the driver and device
- DESCRIPTION** This module implements the Fujitsu MB86960 NICE Ethernet network interface driver. This driver is non-generic and has only been run on the Fujitsu SPARClite Evaluation Board. It currently supports only unit number zero. The driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below.
- BOARD LAYOUT** This device is on-board. No jumpering diagram is necessary.
- EXTERNAL INTERFACE**
This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine; there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the initialization routine is NULL.
The only user-callable routine is `fnattach()`, which publishes the `fn` interface and initializes the driver and device.
- TARGET-SPECIFIC PARAMETERS**
External support routines provide all parameters:
- device I/O address
This parameter specifies the base address of the device's I/O register set. This address is assumed to live in SPARClite alternate address space.
 - interrupt vector
This parameter specifies the interrupt vector to be used by the driver to service an interrupt from the NICE device. The driver will connect the interrupt handler to this vector by calling `intConnect()`.
 - Ethernet address
This parameter specifies the unique, six-byte address assigned to the VxWorks target on the Ethernet.
- EXTERNAL SUPPORT REQUIREMENTS**
This driver requires five external support functions:
- `char *sysEnetIOAddrGet (int unit)`
This routine returns the base address of the NICE control registers. The driver calls

this routine once, using *fnattach()*.

int sysEnetVectGet (int unit)

This routine returns the interrupt vector number to be used to connect the driver's interrupt handler. The driver calls this routine once, using *fnattach()*.

STATUS sysEnetAddrGet (int unit, char *pCopy)

This routine provides the six-byte Ethernet address used by *unit*. It must copy the six-byte address to the space provided by *pCopy*. It returns OK, or ERROR if it fails. The driver calls this routine once, using *fnattach()*.

void sysEnetIntEnable (int unit), void sysEnetIntDisable (int unit)

These routines enable or disable the interrupt from the NICE for the specified *unit*. Typically, this involves interrupt controller hardware, either internal or external to the CPU. The driver calls these routines only during initialization, using *fnattach()*.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- 3944 bytes in text section (text)
- 0 bytes in the initialized data section (data)
- 3152 bytes in the uninitialized data section (BSS)

The above data and BSS requirements are for the SPARClike architecture and may vary for other architectures. Code size (text) varies greatly between architectures and is therefore not quoted here.

The NICE device maintains a private buffer for all packets transmitted and received. Therefore, the driver does not require any system memory to share with the device. This also eliminates all data cache coherency issues.

SEE ALSO

ifLib

if_in

| | |
|-----------------------------------|---|
| NAME | if_in – AMD Am7990 LANCE Ethernet network interface driver |
| ROUTINES | <i>lnattach()</i> – publish the In network interface and initialize driver structures |
| DESCRIPTION | <p>This module implements the Advanced Micro Devices Am7990 LANCE Ethernet network interface driver.</p> <p>This driver is designed to be moderately generic, operating unmodified across the range of architectures and targets supported by VxWorks. To achieve this, the driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below. If any of the assumptions stated below are not true for your particular hardware, this driver will probably not function correctly with it.</p> <p>This driver supports only one LANCE unit per CPU. The driver can be configured to support big-endian or little-endian architectures. It contains error recovery code to handle known device errata related to DMA activity.</p> |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | <p>This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine; there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the initialization routine is NULL.</p> <p>The only user-callable routine is <i>lnattach()</i>, which publishes the In interface and initializes the driver and device.</p> |
| TARGET-SPECIFIC PARAMETERS | <p>bus mode</p> <p>This parameter is a global variable that can be modified at run-time.</p> <p>The LANCE control register #3 determines the bus mode of the device, allowing the support of big-endian and little-endian architectures. This parameter, defined as "u_short InCSR_3B", is the value that will be placed into LANCE control register #3. The default value supports Motorola-type buses. For information about changing this parameter, see the manual <i>Advanced Micro Devices Local Area Network Controller Am7990 (LANCE)</i>.</p> <p>base address of device registers</p> <p>This parameter is passed to the driver by <i>lnattach()</i>. It indicates to the driver where to find the RDP register.</p> |

if_In

The LANCE presents two registers to the external interface, the RDP (register data port) and RAP (register address port) registers. This driver assumes that these two registers occupy two unique addresses in a memory space that is directly accessible by the CPU executing this driver. The driver assumes that the RDP register is mapped at a lower address than the RAP register; the RDP register is therefore considered the "base address."

interrupt vector

This parameter is passed to the driver by *lnattach()*.

This driver configures the LANCE device to generate hardware interrupts for various events within the device; thus it contains an interrupt handler routine. The driver calls *intConnect()* to connect its interrupt handler to the interrupt vector generated as a result of the LANCE interrupt.

interrupt level

This parameter is passed to the driver by *lnattach()*.

Some targets use additional interrupt controller devices to help organize and service the various interrupt sources. This driver avoids all board-specific knowledge of such devices. During the driver's initialization, the external routine *sysLanIntEnable()* is called to perform any board-specific operations required to allow the servicing of a LANCE interrupt. For a description of *sysLanIntEnable()*, see "External Support Requirements" below.

This parameter is passed to the external routine.

shared memory address

This parameter is passed to the driver by *lnattach()*.

The LANCE device is a DMA type of device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the LANCE. It assumes that this shared memory is directly available to it without any arbitration or timing concerns.

This parameter can be used to specify an explicit memory region for use by the LANCE. This should be done on hardware that restricts the LANCE to a particular memory region. The constant NONE can be used to indicate that there are no memory limitations, in which case, the driver attempts to allocate the shared memory from the system space.

shared memory size

This parameter is passed to the driver by *lnattach()*.

This parameter can be used to explicitly limit the amount of shared memory (bytes) this driver will use. The constant NONE can be used to indicate no specific size limitation. This parameter is used only if a specific memory region is provided to the driver.

shared memory width

This parameter is passed to the driver by *lnattach()*.

Some target hardware that restricts the shared memory region to a specific location also restricts the access width to this region by the CPU. On these targets, performing an access of an invalid width will cause a bus error.

This parameter can be used to specify the number of bytes of access width to be used by the driver during access to the shared memory. The constant `NONE` can be used to indicate no restrictions.

Current internal support for this mechanism is not robust; implementation may not work on all targets requiring these restrictions.

Ethernet address

This parameter is obtained directly from a global memory location.

During initialization, the driver needs to know the Ethernet address for the LANCE device. The driver assumes this address is available in a global, six-byte character array, `InEnetAddr[]`. This array is typically created and stuffed by the BSP code.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires one external support function:

void sysLanIntEnable (int level)

This routine provides a target-specific enable of the interrupt for the LANCE device. Typically, this involves interrupt controller hardware, either internal or external to the CPU.

This routine is called once, from the *lnattach()* routine.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- 24 bytes in the initialized data section (data)
- 208 bytes in the uninitialized data section (BSS)

The above data and BSS requirements are for the MC68020 architecture and may vary for other architectures. Code size (text) varies greatly between architectures and is therefore not quoted here.

If the driver is not given a specific region of memory via the *lnattach()* routine, then it calls *cacheDmaMalloc()* to allocate the memory to be shared with the LANCE. The size requested is 80,542 bytes. If a memory region is provided to the driver, the size of this region is adjustable to suit user needs.

The LANCE can only be operated if the shared memory region is write-coherent with the data cache. The driver cannot maintain cache coherency for data that is written by the driver because fields within the shared structures are asynchronously modified by both the driver and the device, and these fields may share the same cache line.

SEE ALSO

ifLib, *Advanced Micro Devices Local Area Network Controller Am7990 (LANCE)*

if_InPci

| | |
|---------------------------|---|
| NAME | if_InPci – AMD Am79C970 PCnet-PCI Ethernet network interface driver |
| ROUTINES | <i>InPciattach()</i> – publish the InPci network interface and initialize the driver and device |
| DESCRIPTION | <p>This module implements the Advanced Micro Devices Am79C970 PCnet-PCI Ethernet 32 bit network interface driver.</p> <p>The PCnet-PCI ethernet controller is inherently little endian because the chip is designed to operate on a PCI bus which is a little endian bus. The software interface to the driver is divided into three parts. The first part is the PCI configuration registers and their set up. This part is done at the BSP level in the various BSPs which use this driver. The second and third part are dealt in the driver. The second part of the interface comprises of the I/O control registers and their programming. The third part of the interface comprises of the descriptors and the buffers.</p> <p>This driver is designed to be moderately generic, operating unmodified across the range of architectures and targets supported by VxWorks. To achieve this, the driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below. If any of the assumptions stated below are not true for your particular hardware, this driver will probably not function correctly with it.</p> <p>This driver supports only one LANCE unit per CPU. The driver can be configured to support big-endian or little-endian architectures. It contains error recovery code to handle known device errata related to DMA activity.</p> <p>Big endian processors can be connected to the PCI bus through some controllers which take care of hardware byte swapping. In such cases all the registers which the chip DMA s to have to be swapped and written to, so that when the hardware swaps the accesses, the chip would see them correctly. The chip still has to be programmed to operated in little endian mode as it is on the PCI bus. If the cpu board hardware automatically swaps all the accesses to and from the PCI bus, then input and output byte stream need not be swapped.</p> |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | <p>This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine; there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the initialization routine is NULL.</p> <p>The only user-callable routine is <i>InPciattach()</i>, which publishes the InPci interface and initializes the driver and device.</p> |

TARGET-SPECIFIC PARAMETERS

bus mode

This parameter is a global variable that can be modified at run-time.

The LANCE control register #3 determines the bus mode of the device, allowing the support of big-endian and little-endian architectures. This parameter, defined as "u_long *InPciCSR_3B*", is the value that will be placed into LANCE control register #3. The default value supports Motorola-type buses. For information about changing this parameter, see the manual *Advanced Micro Devices Local Area Network Controller Am79C970 (PCnet-PCI)*.

base address of device registers

This parameter is passed to the driver by *InPciattach()*. It indicates to the driver where to find the RDP register.

The LANCE presents two registers to the external interface, the RDP (register data port) and RAP (register address port) registers. This driver assumes that these two registers occupy two unique addresses in a memory space that is directly accessible by the CPU executing this driver. The driver assumes that the RDP register is mapped at a lower address than the RAP register; the RDP register is therefore considered the "base address."

interrupt vector

This parameter is passed to the driver by *InPciattach()*.

This driver configures the LANCE device to generate hardware interrupts for various events within the device; thus it contains an interrupt handler routine. The driver calls *intConnect()* to connect its interrupt handler to the interrupt vector generated as a result of the LANCE interrupt.

interrupt level

This parameter is passed to the driver by *InPciattach()*.

Some targets use additional interrupt controller devices to help organize and service the various interrupt sources. This driver avoids all board-specific knowledge of such devices. During the driver's initialization, the external routine *sysLanIntEnable()* is called to perform any board-specific operations required to turn on LANCE interrupt generation. A similar routine, *sysLanIntDisable()*, is called by the driver before a LANCE reset to perform board-specific operations required to turn off LANCE interrupt generation. For a description of *sysLanIntEnable()*, and *sysLanIntDisable()*, see "External Support Requirements" below.

This parameter is passed to the external routine.

shared memory address

This parameter is passed to the driver by *InPciattach()*.

The LANCE device is a DMA type of device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the LANCE. It assumes that this shared

memory is directly available to it without any arbitration or timing concerns.

This parameter can be used to specify an explicit memory region for use by the LANCE. This should be done on hardware that restricts the LANCE to a particular memory region. The constant NONE can be used to indicate that there are no memory limitations, in which case, the driver attempts to allocate the shared memory from the system space.

shared memory size

This parameter is passed to the driver by *InPciattach()*.

This parameter can be used to explicitly limit the amount of shared memory (bytes) this driver will use. The constant NONE can be used to indicate no specific size limitation. This parameter is used only if a specific memory region is provided to the driver.

shared memory width

This parameter is passed to the driver by *InPciattach()*.

Some target hardware that restricts the shared memory region to a specific location also restricts the access width to this region by the CPU. On these targets, performing an access of an invalid width will cause a bus error.

This parameter can be used to specify the number of bytes of access width to be used by the driver during access to the shared memory. The constant NONE can be used to indicate no restrictions.

Current internal support for this mechanism is not robust; implementation may not work on all targets requiring these restrictions.

shared memory buffer size

This parameter is passed to the driver by *InPciattach()*.

The driver and LANCE device exchange network data in buffers. This parameter permits the size of these individual buffers to be limited. A value of zero indicates that the default buffer size should be used. The default buffer size is large enough to hold a maximum-size Ethernet packet.

Use of this parameter should be rare. Network performance will be affected, since the target will no longer be able to receive all valid packet sizes.

Ethernet address

This parameter is obtained directly from a global memory location.

During initialization, the driver needs to know the Ethernet address for the LANCE device. The driver assumes that this address is available in a global, six-byte character array, *InEnetAddr[]*. This array is typically created and stuffed by the BSP code.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires one external support function:

void sysLanIntEnable (int level)

This routine provides a target-specific enable of the interrupt for the LANCE device. Typically, this involves programming an interrupt controller hardware, either internal or external to the CPU.

This routine is called during chip initialization, at startup and each LANCE device reset.

void sysLanIntDisable (int level)

This routine provides a target-specific disable of the interrupt for the LANCE device. Typically, this involves programming an interrupt controller hardware, either internal or external to the CPU.

This routine is called before a LANCE device reset.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- 24 bytes in the initialized data section (data)
- 208 bytes in the uninitialized data section (BSS)

The above data and BSS requirements are for the MC68020 architecture and may vary for other architectures. Code size (text) varies greatly between architectures and is therefore not quoted here.

If the driver is not given a specific region of memory via the *InPciattach()* routine, then it calls *cacheDmaMalloc()* to allocate the memory to be shared with the LANCE. The size requested is 80,542 bytes. If a memory region is provided to the driver, the size of this region is adjustable to suit user needs.

The LANCE can only be operated if the shared memory region is write-coherent with the data cache. The driver cannot maintain cache coherency for the device for data that is written by the driver because fields within the shared structures are asynchronously modified by both the driver and the device, and these fields may share the same cache line.

SEE ALSO

ifLib, *Advanced Micro Devices PCnet-PCI Ethernet Controller for PCI*.

if_loop

| | |
|---------------------|--|
| NAME | if_loop – software loopback network interface driver |
| ROUTINES | <i>loattach()</i> – publish the lo network interface and initialize the driver and pseudo-device |
| DESCRIPTION | <p>This module implements the software loopback network interface driver. The only user-callable routine is <i>loattach()</i>, which publishes the lo interface and initializes the driver and device.</p> <p>This interface is used for protocol testing and timing. By default, the loopback interface is accessible at Internet address 127.0.0.1.</p> |
| BOARD LAYOUT | This device is "software only." A jumpering diagram is not applicable. |
| SEE ALSO | ifLib |

if_mbc

| | |
|--------------------|--|
| NAME | if_mbc – Motorola 68EN302 network-interface driver |
| ROUTINES | <i>mbcattach()</i> – publish the mbc network interface and initialize the driver <i>mbcStartOutput()</i> – output packet to network interface device <i>mbcIntr()</i> – network interface interrupt handler |
| DESCRIPTION | <p>This is a driver for the Ethernet controller on the 68EN302 chip. The device supports a 16-bit interface, data rates up to 10 Mbps, a dual-ported RAM, and transparent DMA. The dual-ported RAM is used for a 64-entry CAM table, and a 128-entry buffer descriptor table. The CAM table is used to set the Ethernet address of the Ethernet device or to program multicast addresses. The buffer descriptor table is partitioned into fixed-size transmit and receive tables. The DMA operation is transparent and transfers data between the internal FIFOs and external buffers pointed to by the receive- and transmit-buffer descriptors during transmits and receives.</p> <p>The driver currently supports one Ethernet module controller, but it can be extended to support multiple controllers when needed. An Ethernet module is initialized by calling <i>mbcattach()</i>.</p> <p>The driver supports buffer loaning for performance and input/output hook routines. It does not support multicast addresses.</p> |

The driver requires that the memory used for transmit and receive buffers be allocated in cache-safe RAM area.

A glitch in the EN302 Rev 0.1 device causes the Ethernet transmitter to lock up from time to time. The driver uses a watchdog timer to reset the Ethernet device when the device runs out of transmit buffers and cannot recover within 20 clock ticks.

BOARD LAYOUT This device is on-chip. No jumpering diagram is necessary.

EXTERNAL INTERFACE

This driver presents the standard WRS network driver API: first the device unit must be attached with the *mbcattach()* routine, then it must be initialized with the *mbcInit()* routine.

The only user-callable routine is *mbcattach()*, which publishes the **mbc**interface and initializes the driver structures.

TARGET-SPECIFIC PARAMETERS

Ethernet module base address

This parameter is passed to the driver via *mbcattach()*.

This parameter is the base address of the Ethernet module. The driver addresses all other Ethernet device registers as offsets from this address.

interrupt vector number

This parameter is passed to the driver via *mbcattach()*.

The driver configures the Ethernet device to use this parameter while generating interrupt ack cycles. The interrupt service routine *mbcIntr()* is expected to be attached to the corresponding interrupt vector externally, typically in *sysHwInit2()*.

number of transmit and receive buffer descriptors

These parameters are passed to the driver via *mbcattach()*.

The number of transmit and receive buffer descriptors (BDs) used is configurable by the user while attaching the driver. Each BD is 8 bytes in size and resides in the chip's dual-ported memory, while its associated buffer, 1520 bytes in size, resides in cache-safe conventional RAM. A minimum of 2 receive and 2 transmit BDs should be allocated. If this parameter is NULL, a default of 32 BDs will be used. The maximum number of BDs depends on how the dual-ported BD RAM is partitioned. The 128 BDs in the dual-ported BD RAM can be partitioned into transmit and receive BD regions with 8, 16, 32, or 64 transmit BDs and corresponding 120, 112, 96, or 64 receive BDs.

Ethernet DMA parameters

This parameter is passed to the driver via *mbcattach()*.

This parameter is used to specify the settings of burst limit, water-mark, and transmit early, which control the Ethernet DMA, and is used to set the EDMA register.

base address of the buffer pool

This parameter is passed to the driver via *mbcattach()*.

This parameter is used to notify the driver that space for the transmit and receive buffers need not be allocated, but should be taken from a cache-coherent private memory space provided by the user at the given address. The user should be aware that memory used for buffers must be 4-byte aligned and non-cacheable. All the buffers must fit in the given memory space; no checking will be performed. This includes all transmit and receive buffers (see above) and an additional 16 receive loaner buffers, unless the number of receive BDs is less than 16, in which case that number of loaner buffers will be used. Each buffer is 1520 bytes. If this parameter is "NONE", space for buffers will be obtained by calling *cacheDmaMalloc()* in *cpmattach()*.

EXTERNAL SUPPORT REQUIREMENTS

The driver requires the following support functions:

STATUS *sysEnetAddrGet* (int unit, UINT8 * addr)

The driver expects this routine to provide the six-byte Ethernet hardware address that will be used by *unit*. This routine must copy the six-byte address to the space provided by *addr*. This routine is expected to return OK on success, or ERROR. The driver calls this routine, during device initialization, from the *cpmInit()* routine.

SYSTEM RESOURCE USAGE

The driver requires the following system resource:

- one mutual exclusion semaphore
- one interrupt vector
- one watchdog timer
- 0 bytes in the initialized data section (data)
- 296 bytes in the uninitialized data section (bss)

The data and BSS sections are quoted for the CPU32 architecture.

If the driver allocates the memory shared with the Ethernet device unit, it does so by calling the *cacheDmaMalloc()* routine. For the default case of 32 transmit buffers, 32 receive buffers, and 16 loaner buffers, the total size requested is 121,600 bytes. If a non-cacheable memory region is provided by the user, the size of this region should be this amount, unless the user has specified a different number of transmit or receive BDs.

This driver can only operate if the shared memory region is non-cacheable, or if the hardware implements bus snooping. The driver cannot maintain cache coherency for the device because the buffers are asynchronously modified by both the driver and the device, and these fields may share the same cache line. Additionally, the chip's dual-ported RAM must be declared as non-cacheable memory where applicable.

SEE ALSO

ifLib, *Motorola MC68EN302 User's Manual, Motorola MC68EN302 Device Errata, May 30, 1996*

if_nicEvb

| | |
|--------------------------------------|--|
| NAME | if_nicEvb – National Semiconductor ST-NIC Chip network interface driver |
| ROUTINES | <i>nicEvbattach()</i> – publish and initialize the nicEvb network interface driver <i>nicTxStartup()</i> – the driver’s actual output routine |
| DESCRIPTION | <p>This module implements the National Semiconductor 83902A ST-NIC Ethernet network interface driver.</p> <p>This driver is non-generic and is for use on the IBM EVB403 board. Only unit number zero is supported. The driver must be given several target-specific parameters. These parameters, and the mechanisms used to communicate them to the driver, are detailed below.</p> |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | <p>This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine; there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the initialization routine is NULL.</p> <p>The only user-callable routine is <i>nicEvbattach()</i>, which publishes the nicEvb interface and initializes the driver and device.</p> |
| TARGET-SPECIFIC PARAMETERS | <p>device I/O address This parameter is passed to the driver by <i>nicEvbattach()</i>. It specifies the base address of the device’s I/O register set.</p> <p>interrupt vector This parameter is passed to the driver by <i>nicEvbattach()</i>. It specifies the interrupt vector to be used by the driver to service an interrupt from the ST-NIC device. The driver will connect the interrupt handler to this vector by calling <i>intConnect()</i>.</p> <p>device restart/reset delay The global variable <i>nicRestartDelay</i> (UINT32), defined in this file, should be initialized in the BSP <i>sysHwInit()</i> routine. <i>nicRestartDelay</i> is used only with PowerPC platform and is equal to the number of time base increments which makes for 1.6 msec. This corresponds to the delay necessary to respect when restarting or resetting the device.</p> |
| EXTERNAL SUPPORT REQUIREMENTS | The driver requires the following support functions: |

if_sl

STATUS `sysEnetAddrGet (int unit, UINT8 * addr)`

The driver expects this routine to provide the six-byte Ethernet hardware address that will be used by *unit*. This routine must copy the six-byte address to the space provided by *addr*. This routine is expected to return OK on success, or ERROR. The driver calls this routine, during device initialization, from the *nicEnetAddrGet()* routine.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector

SEE ALSO

`ifLib`

if_sl

NAME

`if_sl` – Serial Line IP (SLIP) network interface driver

ROUTINES

slipInit() – initialize a SLIP interface
slipBaudSet() – set the baud rate for a SLIP interface
slattach() – publish the `sl` network interface and initialize the driver and device
slipDelete() – delete a SLIP interface

DESCRIPTION

This module implements the VxWorks Serial Line IP (SLIP) network interface driver. Support for compressed TCP/IP headers (CSLIP) is included.

The SLIP driver enables VxWorks to talk to other machines over serial connections by encapsulating IP packets into streams of bytes suitable for serial transmission.

USER-CALLABLE ROUTINES

SLIP devices are initialized using *slipInit()*. Its parameters specify the Internet address for both sides of the SLIP point-to-point link, the name of the tty device on the local host, and options to enable CSLIP header compression. The *slipInit()* routine calls *slattach()* to attach the SLIP interface to the network. The *slipDelete()* routine deletes a specified SLIP interface.

LINK-LEVEL PROTOCOL

SLIP is a simple protocol that uses four token characters to delimit each packet:

- END (0300)
- ESC (0333)
- TRANS_END (0334)
- TRANS_ESC (0335)

The END character denotes the end of an IP packet. The ESC character is used with `TRANS_END` and `TRANS_ESC` to circumvent potential occurrences of END or ESC within a packet. If the END character is to be embedded, SLIP sends "ESC `TRANS_END`" to avoid confusion between a SLIP-specific END and actual data whose value is END. If the ESC character is to be embedded, then SLIP sends "ESC `TRANS_ESC`" to avoid confusion. (Note that the SLIP ESC is not the same as the ASCII ESC.)

On the receiving side of the connection, SLIP uses the opposite actions to decode the SLIP packets. Whenever an END character is received, SLIP assumes a full IP packet has been received and sends it up to the IP layer.

TARGET-SPECIFIC PARAMETERS

The global flag `slipLoopBack` is set to 1 by default. This flag enables the packets to be sent to the loopback interface if they are destined to to a local slip interface address. By setting this flag, any packets sent to a local slip interface address will not be seen on the actual serial link. Set this flag to 0 to turn off this facility. If this flag is not set any packets sent to the local slip interface address will actually be sent out on the link and it is the peer's responsibility to loop the packet back.

IMPLEMENTATION The write side of a SLIP connection is an independent task. Each SLIP interface has its own output task that sends SLIP packets over a particular tty device channel. Whenever a packet is ready to be sent out, the SLIP driver activates this task by giving a semaphore. When the semaphore is available, the output task performs packetization (as explained above) and writes the packet to the tty device.

The receiving side is implemented as a "hook" into the tty driver. A tty `ioctl()` request, `FIOPROTOHOOK`, informs the tty driver to call the SLIP interrupt routine every time a character is received from a serial port. By tracking the number of characters and watching for the END character, the number of calls to `read()` and context switching time have been reduced. The SLIP interrupt routine will queue a call to the SLIP read routine only when it knows that a packet is ready in the tty driver's ring buffer. The SLIP read routine will read a whole SLIP packet at a time and process it according to the SLIP framing rules. When a full IP packet is decoded out of a SLIP packet, it is queued to IP's input queue.

CSLIP compression is implemented to decrease the size of the TCP/IP header information, thereby improving the data to header size ratio. CSLIP manipulates header information just before a packet is sent and just after a packet is received. Only TCP/IP headers are compressed and uncompressed; other protocol types are sent and received normally. A functioning CSLIP driver is required on the peer (destination) end of the physical link in order to carry out a CSLIP "conversation."

Multiple units are supported by this driver. Each individual unit may have CSLIP support disabled or enabled, independent of the state of other units.

BOARD LAYOUT No hardware is directly associated with this driver; therefore, a jumpering diagram is not applicable.

if_sm

SEE ALSO **ifLib, tyLib**, John Romkey: RFC-1055, *A Nonstandard for Transmission of IP Datagrams Over Serial Lines: SLIP*, Van Jacobson: RFC-1144, entitled *Compressing TCP/IP Headers for Low-Speed Serial Links*

ACKNOWLEDGEMENT

This program is based on original work done by Rick Adams of The Center for Seismic Studies and Chris Torek of The University of Maryland. The CSLIP enhancements are based on work done by Van Jacobson of University of California, Berkeley for the "cslip-2.7" release.

if_sm

NAME **if_sm** – shared memory backplane network interface driver

ROUTINES **smIfAttach()** – publish the **sm** interface and initialize the driver and device

DESCRIPTION This module implements the VxWorks shared memory backplane network interface driver.

This driver is designed to be moderately generic, operating unmodified across the range of hosts and targets supported by VxWorks. To achieve this, the driver must be given several target-specific parameters, and some external support routines must be provided. These parameters are detailed below.

The only user-callable routine is **smIfAttach()**, which publishes the **sm** interface and initializes the driver and device.

This driver is layered between the shared memory packet library and the network modules. The backplane driver gives CPUs residing on a common backplane the ability to communicate using IP (via shared memory).

This driver is used both under VxWorks and other host operating systems, e.g., SunOs.

BOARD LAYOUT This device is "software only." There is no jumpering diagram required.

TARGET-SPECIFIC PARAMETERS

local address of anchor

This parameter is passed to the driver by **smIfAttach()**. It is the local address by which the local CPU accesses the shared memory anchor.

maximum number of input packets

This parameter is passed to the driver by **smIfAttach()**. It specifies the maximum number of incoming shared memory packets that can be queued to this CPU at one time.

method of notification

These parameters are passed to the driver by *smIfAttach()*. Four parameters can be used to allow a CPU to announce the method by which it is to be notified of input packets that have been queued to it.

heartbeat frequency

This parameter is passed to the driver by *smIfAttach()*. It specifies the frequency of the shared memory anchor's heartbeat, which is expressed in terms of the number of CPU ticks on the local CPU corresponding to one heartbeat period.

number of buffers to loan

This parameter is passed to the driver by *smIfAttach()*. When the value is non-zero, this parameter specifies the number of shared memory packets available to be loaned out.

SEE ALSO **ifLib, smNetLib**

if_sn

NAME **if_sn** – National Semiconductor DP83932B SONIC Ethernet network driver

ROUTINES *snattach()* – publish the **sn** network interface and initialize the driver and device

DESCRIPTION This module implements the National Semiconductor DP83932 SONIC Ethernet network interface driver.

This driver is designed to be moderately generic, operating unmodified across the range of architectures and targets supported by VxWorks. To achieve this, the driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below. If any of the assumptions stated below are not true for your particular hardware, this driver will probably not function correctly with it. This driver supports up to four individual units per CPU.

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach routine; there is no separate initialization routine. Therefore, in the global interface structure, the function pointer to the initialization routine is NULL.

There is one user-callable routine, *snattach()*; for details, see the manual entry for this routine.

TARGET-SPECIFIC PARAMETERS

device I/O address

This parameter is passed to the driver by *snattach()*. It specifies the base address of the device's I/O register set.

interrupt vector

This parameter is passed to the driver by *snattach()*. It specifies the interrupt vector to be used by the driver to service an interrupt from the SONIC device. The driver will connect the interrupt handler to this vector by calling *intConnect()*.

Ethernet address

This parameter is obtained by calling an external support routine. It specifies the unique, six-byte address assigned to the VxWorks target on the Ethernet.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires five external support functions:

void sysEnetInit (int unit)

This routine performs any target-specific operations that must be executed before the SONIC device is initialized. The driver calls this routine, once per unit, from *snattach()*.

STATUS sysEnetAddrGet (int unit, char *pCopy)

This routine provides the six-byte Ethernet address used by *unit*. It must copy the six-byte address to the space provided by *pCopy*. This routine returns OK, or ERROR if it fails. The driver calls this routine, once per unit, from *snattach()*.

void sysEnetIntEnable (int unit), void sysEnetIntDisable (int unit)

These routines enable or disable the interrupt from the SONIC device for the specified *unit*. Typically, this involves interrupt controller hardware, either internal or external to the CPU. The driver calls these routines only during initialization, from *snattach()*.

void sysEnetIntAck (int unit)

This routine performs any interrupt acknowledgement or clearing that may be required. This typically involves an operation to some interrupt control hardware. The driver calls this routine from the interrupt handler.

DEVICE CONFIGURATION

Two global variables, **snDcr** and **snDcr2**, are used to set the SONIC device configuration registers. By default, the device is programmed in 32-bit mode with zero wait states. If these values are not suitable, the **snDcr** and **snDcr2** variables should be modified before calling *snattach()*. See the SONIC manual to change these parameters.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one interrupt vector
- 0 bytes in the initialized data section (data)
- 696 bytes in the uninitialized data section (BSS)

The above data and BSS requirements are for the MC68020 architecture and may vary for other architectures. Code size (text) varies greatly between architectures and is therefore not quoted here.

This driver uses *cacheDmaMalloc()* to allocate the memory to be shared with the SONIC device. The size requested is 117,188 bytes.

The SONIC device can only be operated if the shared memory region is write-coherent with the data cache. The driver cannot maintain cache coherency for the device for data that is written by the driver because fields within the shared structures are asynchronously modified by the driver and the device, and these fields may share the same cache line.

NOTE 1 The previous transmit descriptor does not exist until the transmitter has been asked to send at least one packet. Unfortunately the test for this condition must be done every time a new descriptor is to be added, even though the condition is only true the first time. However, it is a valuable test, since we should not use the fragment count field as an index if it is 0.

NOTE 2 There are some things unsupported in this version:

- a) buffer loaning on receive
- b) output hooks
- c) trailer protocol
- d) promiscuous mode

Also, the receive setup needs work so that the number of RRA descriptors is not fixed at four. It would be a nice addition to allow all the sizes of the shared memory structures to be specified by the runtime functions that call our init routines.

SEE ALSO **ifLib**

if_ulip

| | |
|--------------------|--|
| NAME | if_ulip – network interface driver for User Level IP (VxSim) |
| ROUTINES | <p><i>ulipInit()</i> – initialize the ULIP interface (VxSim)</p> <p><i>ulattach()</i> – attach a ULIP interface to a list of network interfaces (VxSim)</p> <p><i>ulipDelete()</i> – delete a ULIP interface (VxSim)</p> <p><i>ulStartOutput()</i> – push packets onto "interface"</p> <p><i>ulipDebugSet()</i> – Set debug flag in UNIX's ULIP driver</p> |
| DESCRIPTION | <p>This module implements the VxWorks User Level IP (ULIP) network driver. The ULIP driver allows VxWorks under UNIX to talk to other machines by handing off IP packets to the UNIX host for processing.</p> <p>The ULIP driver is automatically included and initialized by the VxSim BSPs; normally there is no need for applications to use these routines directly.</p> |

USER-CALLABLE ROUTINES

When initializing the device, it is necessary to specify the Internet address for both sides of the ULIP point-to-point link (local side and the remote side) using *ulipInit()*.

```

STATUS ulipInit
(
    int unit,          /* ULIP unit number (0 - NULIP-1) */
    char *myAddr,     /* IP address of the interface */
    char *peerAddr,   /* IP address of the remote peer interface */
    int procnum       /* processor number to map to ULIP interface */
)

```

For example, the following initializes a ULIP device whose Internet address is 127.0.1.1:

```
ulipInit (0, "127.0.1.1", "147.11.1.132", 1);
```

The standard network interface call is:

```

STATUS ulattach
(
    int unit /* unit number */
)

```

However, it should not be called. The following call will delete the first ULIP interface from the list of network interfaces:

```
ulipDelete (0); /* unit number */
```

Up to NULIP(2) units may be created.

SEE ALSO *VxWorks Programmer's Guide: VxSim*

if_ultra

| | |
|----------------------|---|
| NAME | if_ultra – SMC Elite Ultra Ethernet network interface driver |
| ROUTINES | <i>ultraattach()</i> – publish ultra interface and initialize device <i>ultraPut()</i> – copy a packet to the interface. <i>ultraShow()</i> – display statistics for the ultra network interface |
| DESCRIPTION | This module implements the SMC Elite Ultra Ethernet network interface driver. This driver supports single transmission and multiple reception. The Current register is a write pointer to the ring. The Bound register is a read pointer from the ring. This driver gets the Current register at the interrupt level and sets the Bound register at the task level. The interrupt is never masked at the task level. |
| CONFIGURATION | The W1 jumper should be set in the position of "Software Configuration". The defined I/O address in config.h must match the one stored in EEROM. The RAM address, the RAM size, and the IRQ level are defined in config.h . IRQ levels 2,3,5,7,10,11,15 are supported. |

EXTERNAL INTERFACE

The only user-callable routines are *ultraattach()* and *ultraShow()*:

ultraattach()

publishes the **ultra** interface and initializes the driver and device.

ultraShow()

displays statistics that are collected in the interrupt handler.

igmpShow

| | |
|--------------------|--|
| NAME | igmpShow – IGMP information display routines |
| ROUTINES | <i>igmpShowInit()</i> – initialize IGMP show routines <i>igmpstatShow()</i> – display statistics for IGMP |
| DESCRIPTION | This library provides routines to show IGMP related statistics. Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books: <ul style="list-style-type: none">– <i>TCP/IP Illustrated Volume II, The Implementation</i>, by Richard Stevens– <i>The Design and Implementation of the 4.4 BSD UNIX Operating System</i>, by Leffler, |

McKusick, Karels and Quarterman

The *igmpShowInit()* routine links the IGMP show facility into the VxWorks system. This is performed automatically if `INCLUDE_NET_SHOW` is defined in `configAll.h`.

SEE ALSO `netLib`, `netShow`, *Network Programmer's Guide*

inetLib

NAME `inetLib` – Internet address manipulation routines

ROUTINES

- inet_addr()* – convert a dot notation Internet address to a long integer
- inet_lnaof()* – get the local address (host number) from the Internet address
- inet_makeaddr_b()* – form an Internet address from network and host numbers
- inet_makeaddr()* – form an Internet address from network and host numbers
- inet_netof()* – return the network number from an Internet address
- inet_netof_string()* – extract the network address in dot notation
- inet_network()* – convert an Internet network number from string to address
- inet_ntoa_b()* – convert a network address to dot notation, store it in a buffer
- inet_ntoa()* – convert a network address to dotted decimal notation
- inet_aton()* – convert a network address from dot notation, store in a structure

DESCRIPTION This library provides routines for manipulating Internet addresses, including the UNIX BSD 4.3 `inet_` routines. It includes routines for converting between character addresses in Internet standard dotted decimal notation and integer addresses, routines for extracting the network and host portions out of an Internet address, and routines for constructing Internet addresses given the network and host address parts.

All Internet addresses are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES

Internet addresses are typically specified in dotted decimal notation or as a 4-byte number. Values specified using the dotted decimal notation take one of the following forms:

```
a.b.c.d
a.b.c
a.b
a
```

If four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is

viewed as a 32-bit integer quantity on any MC68000 family machine, the bytes referred to above appear as "a.b.c.d" and are ordered from left to right.

If a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right-most two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as "128.net.host".

If a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right-most three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as "net.host".

If only one part is given, the value is stored directly in the network address without any byte rearrangement.

Although dotted decimal notation is the default, it is possible to use the dot notation with hexadecimal or octal numbers. The base is indicated using the same prefixes as are used in C. That is, a leading 0x or 0X indicates a hexadecimal number. A leading 0 indicates an octal number. If there is no prefix, the number is interpreted as decimal.

INCLUDE FILES **inetLib.h, inet.h**

SEE ALSO UNIX BSD 4.3 manual entry for inet(3N), *VxWorks Programmer's Guide: Network*

inflateLib

NAME **inflateLib** – inflate code using public domain zlib functions

ROUTINES *inflate()* – inflate compressed code

DESCRIPTION This library is used to inflate a compressed data stream, primarily for boot ROM decompression. Compressed boot ROMs contain a compressed executable in the data segment between the symbols **binArrayStart** and **binArrayEnd** (the compressed data is generated by **deflate** and **binToAsm**). The boot ROM startup code (in **target/src/config/all/bootInit.c**) calls *inflate()* to decompress the executable and then jump to it.

This library is based on the public domain zlib code, which has been modified by Wind River Systems. For more information, see the zlib home page at <http://quest.jpl.nasa.gov/zlib/>.

intArchLib

| | |
|--------------------------------------|---|
| NAME | intArchLib – architecture-dependent interrupt library |
| ROUTINES | <p><i>intLevelSet()</i> – set the interrupt level (MC680x0, SPARC, i960, x86, ARM) <i>intLock()</i> – lock out interrupts <i>intUnlock()</i> – cancel interrupt locks <i>intEnable()</i> – enable corresponding interrupt bits (MIPS, PowerPC, ARM) <i>intDisable()</i> – disable corresponding interrupt bits (MIPS, PowerPC, ARM) <i>intCRGet()</i> – read the contents of the cause register (MIPS) <i>intCRSet()</i> – write the contents of the cause register (MIPS) <i>intSRGet()</i> – read the contents of the status register (MIPS) <i>intSRSet()</i> – update the contents of the status register (MIPS) <i>intConnect()</i> – connect a C routine to a hardware interrupt <i>intHandlerCreate()</i> – construct ISR for a C routine (MC680x0, SPARC, i960, x86, MIPS) <i>intLockLevelSet()</i> – set current interrupt lock-out level (MC680x0, SPARC, i960, x86, ARM) <i>intLockLevelGet()</i> – get current interrupt lock-out level (MC680x0, SPARC, i960, x86, ARM) <i>intVecBaseSet()</i> – set vector (trap) base address (MC680x0, SPARC, i960, x86, MIPS, ARM) <i>intVecBaseGet()</i> – get vector (trap) base address (MC680x0, SPARC, i960, x86, MIPS, ARM) <i>intVecSet()</i> – set a CPU vector (trap) (MC680x0, SPARC, i960, x86, MIPS) <i>intVecGet()</i> – get an interrupt vector (MC680x0, SPARC, i960, x86, MIPS) <i>intVecTableWriteProtect()</i> – write-protect exception vector table (MC680x0, SPARC, i960, x86, ARM) <i>intUninitVecSet()</i> – set the uninitialized vector handler (ARM)</p> |
| DESCRIPTION | <p>This library provides architecture-dependent routines to manipulate and connect to hardware interrupts. Any C language routine can be connected to any interrupt by calling <i>intConnect()</i>. Vectors can be accessed directly by <i>intVecSet()</i> and <i>intVecGet()</i>. The vector (trap) base register (if present) can be accessed by the routines <i>intVecBaseSet()</i> and <i>intVecBaseGet()</i>.</p> <p>Tasks can lock and unlock interrupts by calling <i>intLock()</i> and <i>intUnlock()</i>. The lock-out level can be set and reported by <i>intLockLevelSet()</i> and <i>intLockLevelGet()</i> (MC680x0, SPARC, i960, i386/i486 and ARM only). The routine <i>intLevelSet()</i> changes the current interrupt level of the processor (MC680x0, SPARC, i960 and ARM).</p> |
| WARNING | Do not call VxWorks system routines with interrupts locked. Violating this rule may re-enable interrupts unpredictably. |
| INTERRUPT VECTORS AND NUMBERS | Most of the routines in this library take an interrupt vector as a parameter, which is generally the byte offset into the vector table. Macros are provided to convert between interrupt vectors and interrupt numbers: |

IVEC_TO_INUM (intVector)
converts a vector to a number.

INUM_TO_IVEC (intNumber)
converts a number to a vector.

TRAPNUM_TO_IVEC (trapNumber)
converts a trap number to a vector.

EXAMPLE

To switch between one of several routines for a particular interrupt, the following code fragment is one alternative:

```
vector = INUM_TO_IVEC(some_int_vec_num);
oldfunc = intVecGet (vector);
newfunc = intHandlerCreate (routine, parameter);
intVecSet (vector, newfunc);
...
intVecSet (vector, oldfunc);    /* use original routine */
...
intVecSet (vector, newfunc);    /* reconnect new routine */
```

INCLUDE FILES iv.h, intLib.h

SEE ALSO intLib

intLib

NAME intLib – architecture-independent interrupt subroutine library

ROUTINES *intContext()* – determine if the current state is in interrupt or task context
intCount() – get the current interrupt nesting depth

DESCRIPTION This library provides generic routines for interrupts. Any C language routine can be connected to any interrupt (trap) by calling *intConnect()*, which resides in **intArchLib**. The *intCount()* and *intContext()* routines are used to determine whether the CPU is running in an interrupt context or in a normal task context. For information about architecture-dependent interrupt handling, see the manual entry for **intArchLib**.

INCLUDE FILES intLib.h

SEE ALSO **intArchLib**, *VxWorks Programmer's Guide: Basic OS*

ioLib

| | |
|-------------------------|---|
| NAME | ioLib – I/O interface library |
| ROUTINES | <p><i>creat()</i> – create a file</p> <p><i>unlink()</i> – delete a file (POSIX)</p> <p><i>remove()</i> – remove a file (ANSI)</p> <p><i>open()</i> – open a file</p> <p><i>close()</i> – close a file</p> <p><i>rename()</i> – change the name of a file</p> <p><i>read()</i> – read bytes from a file or device</p> <p><i>write()</i> – write bytes to a file</p> <p><i>ioctl()</i> – perform an I/O control function</p> <p><i>lseek()</i> – set a file read/write pointer</p> <p><i>ioDefPathSet()</i> – set the current default path</p> <p><i>ioDefPathGet()</i> – get the current default path</p> <p><i>chdir()</i> – set the current default path</p> <p><i>getcwd()</i> – get the current default path (POSIX)</p> <p><i>getwd()</i> – get the current default path</p> <p><i>ioGlobalStdSet()</i> – set the file descriptor for global standard input/output/error</p> <p><i>ioGlobalStdGet()</i> – get the file descriptor for global standard input/output/error</p> <p><i>ioTaskStdSet()</i> – set the file descriptor for task standard input/output/error</p> <p><i>ioTaskStdGet()</i> – get the file descriptor for task standard input/output/error</p> <p><i>isatty()</i> – return whether the underlying driver is a tty device</p> |
| DESCRIPTION | <p>This library contains the interface to the basic I/O system. It includes:</p> <ul style="list-style-type: none"> – Interfaces to the seven basic driver-provided functions: <i>creat()</i>, <i>remove()</i>, <i>open()</i>, <i>close()</i>, <i>read()</i>, <i>write()</i>, and <i>ioctl()</i>. – Interfaces to several file system functions, including <i>rename()</i> and <i>lseek()</i>. – Routines to set and get the current working directory. – Routines to assign task and global standard file descriptors. |
| FILE DESCRIPTORS | <p>At the basic I/O level, files are referred to by a file descriptor. A file descriptor is a small integer returned by a call to <i>open()</i> or <i>creat()</i>. The other basic I/O calls take a file descriptor as a parameter to specify the intended file.</p> <p>Three file descriptors are reserved and have special meanings:</p> <ul style="list-style-type: none"> 0 (STD_IN) – standard input 1 (STD_OUT) – standard output 2 (STD_ERR) – standard error output |

VxWorks allows two levels of redirection. First, there is a global assignment of the three standard file descriptors. By default, new tasks use this global assignment. The global assignment of the three standard file descriptors is controlled by the routines *ioGlobalStdSet()* and *ioGlobalStdGet()*.

Second, individual tasks may override the global assignment of these file descriptors with their own assignments that apply only to that task. The assignment of task-specific standard file descriptors is controlled by the routines *ioTaskStdSet()* and *ioTaskStdGet()*.

INCLUDE FILES **ioLib.h**

SEE ALSO **ioLib**, **ansiStdio**, *VxWorks Programmer's Guide: I/O System*

iOlicomEnd

NAME **iOlicomEnd** – END style Intel Olicom PCMCIA network interface driver

ROUTINES *iOlicomEndLoad()* – initialize the driver and device
iOlicomIntHandle() – interrupt service for card interrupts

DESCRIPTION This module implements the Olicom (Intel 82595TX) network interface driver. The physical device is a PCMCIA card. This driver also houses code to manage a Vadem PCMCIA Interface controller on the ARM PID board, which is strictly a subsystem in it's own right.

This network interface driver does not include support for trailer protocols or data chaining. However, buffer loaning has been implemented in an effort to boost performance.

This driver maintains cache coherency by allocating buffer space using the *cacheDmaMalloc()* routine.

BOARD LAYOUT The device resides on a PCMCIA card and is soft configured. No jumpering diagram is necessary.

EXTERNAL INTERFACE

This driver provides the END external interface with the following exceptions. The only external interface is the *iOlicomEndLoad()* routine. All of the paramters are passed as strings in a colon (:) separated list to the load function as an *initString*. The *iOlicomEndLoad()* function uses *strtok()* to parse the string.

The string contains the target specific parameters like this:

```
"io_baseA:attr_baseA:mem_baseA:io_baseB:attr_baseB:mem_baseB: \  
ctrl_base:intVectA:intLevelA:intVectB:intLevelB: \  
txBdNum:rxBdNum:pShMem:shMemSize"
```

TARGET-SPECIFIC PARAMETERS

I/O base address A

This is the first parameter passed to the driver init string. This parameter indicates the base address of the PCMCIA I/O space for socket A.

Attribute base address A

This is the second parameter passed to the driver init string. This parameter indicates the base address of the PCMCIA attribute space for socket A. On the PID board, this should be the offset of the beginning of the attribute space from the beginning of the memory space.

Memory base address A

This is the third parameter passed to the driver init string. This parameter indicates the base address of the PCMCIA memory space for socket A.

I/O base address B

This is the fourth parameter passed to the driver init string. This parameter indicates the base address of the PCMCIA I/O space for socket B.

Attribute base address B

This is the fifth parameter passed to the driver init string. This parameter indicates the base address of the PCMCIA attribute space for socket B. On the PID board, this should be the offset of the beginning of the attribute space from the beginning of the memory space.

Memory base address B

This is the sixth parameter passed to the driver init string. This parameter indicates the base address of the PCMCIA memory space for socket B.

PCMCIA controller base address

This is the seventh parameter passed to the driver init string. This parameter indicates the base address of the Vadem PCMCIA controller.

interrupt vectors and levels

These are the eighth, ninth, tenth and eleventh parameters passed to the driver init string.

The mapping of IRQs generated at the Card/PCMCIA level to interrupt levels and vectors is system dependent. Furthermore the slot holding the PCMCIA card is not initially known. The interrupt levels and vectors for both socket A and socket B must be passed to *iOlicomEndLoad()*, allowing the driver to select the required parameters later.

number of transmit and receive buffer descriptors

These are the twelfth and thirteenth parameters passed to the driver init string.

The number of transmit and receive buffer descriptors (BDs) used is configurable by the user upon attaching the driver. There must be a minimum of two transmit and two receive BDs, and there is a maximum of twenty transmit and twenty receive BDs. If this parameter is "NULL" a default value of 16 BDs will be used.

offset

This is the fourteenth parameter passed to the driver in the init string.

This parameter defines the offset which is used to solve alignment problem.

base address of buffer pool

This is the fifteenth parameter passed to the driver in the init string.

This parameter is used to notify the driver that space for the transmit and receive buffers need not be allocated, but should be taken from a cache-coherent private memory space provided by the user at the given address. The user should be aware that memory used for buffers must be 4-byte aligned and non-cacheable. If this parameter is "NONE", space for buffers will be obtained by calling *cacheDmaMalloc()* in *iOlicomEndLoad()*.

mem size of buffer pool

This is the sixteenth parameter passed to the driver in the init string.

The memory size parameter specifies the size of the pre-allocated memory region. If memory base is specified as NONE (-1), the driver ignores this parameter.

Ethernet address

This parameter is obtained from the Card Information Structure on the Olicom PCMCIA card.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires three external support function:

void sysLanIntEnable (int level)

This routine provides a target-specific interface for enabling Ethernet device interrupts at a specified interrupt level. This routine is called each time that the *iOlicomStart()* routine is called.

void sysLanIntDisable (int level)

This routine provides a target-specific interface for disabling Ethernet device interrupts. The driver calls this routine from the *iOlicomStop()* routine each time a unit is disabled.

void sysBusIntAck(void)

This routine acknowledge the interrupt if it's necessary.

SEE ALSO

muxLib, endLib, *Intel 82595TX ISA/PCMCIA High Integration Ethernet Controller User Manual, Vadem VG-468 PC Card Socket Controller Data Manual.*

ioMmuMicroSparcLib

| | |
|----------------------|--|
| NAME | ioMmuMicroSparcLib – microSparc I/II I/O DMA library |
| ROUTINES | <i>ioMmuMicroSparcInit()</i> – initialize the microSparc I/II I/O MMU data structures <i>ioMmuMicroSparcMap()</i> – map the I/O MMU for microSparc I/II (TMS390S10/MB86904) |
| DESCRIPTION | This library contains the SPARC architecture-specific functions <i>ioMmuMicroSparcInit()</i> and <i>ioMmuMicroSparcMap()</i> , needed to set up the I/O mapping for S-Bus DMA devices using the TI TMS390S10 and the MicroSparc II Mb86904 architecture. |
| INCLUDE FILES | arch/sparc/microSparc.h |
| SEE ALSO | cacheLib, mmuLib, vmLib |

iosLib

| | |
|----------------------|--|
| NAME | iosLib – I/O system library |
| ROUTINES | <i>iosInit()</i> – initialize the I/O system <i>iosDrvInstall()</i> – install an I/O driver <i>iosDrvRemove()</i> – remove an I/O driver <i>iosDevAdd()</i> – add a device to the I/O system <i>iosDevDelete()</i> – delete a device from the I/O system <i>iosDevFind()</i> – find an I/O device in the device list <i>iosFdValue()</i> – validate an open file descriptor and return the driver-specific value |
| DESCRIPTION | <p>This library is the driver-level interface to the I/O system. Its primary purpose is to route user I/O requests to the proper drivers, using the proper parameters. To do this, iosLib keeps tables describing the available drivers (e.g., names, open files).</p> <p>The I/O system should be initialized by calling <i>iosInit()</i>, before calling any other routines in iosLib. Each driver then installs itself by calling <i>iosDrvInstall()</i>. The devices serviced by each driver are added to the I/O system with <i>iosDevAdd()</i>.</p> <p>The I/O system is described more fully in the <i>I/O System</i> chapter of the <i>Programmer's Guide</i>.</p> |
| INCLUDE FILES | iosLib.h |
| SEE ALSO | intLib, ioLib , <i>VxWorks Programmer's Guide: I/O System</i> |

iosShow

| | |
|--------------------|--|
| NAME | iosShow – I/O system show routines |
| ROUTINES | <i>iosShowInit()</i> – initialize the I/O system show facility <i>iosDrvShow()</i> – display a list of system drivers <i>iosDevShow()</i> – display the list of devices in the system <i>iosFdShow()</i> – display a list of file descriptor names in the system |
| DESCRIPTION | This library contains I/O system information display routines. The routine <i>iosShowInit()</i> links the I/O system information show facility into the VxWorks system. It is called automatically when <code>INCLUDE_SHOW_ROUTINES</code> is defined in <code>configAll.h</code> . |
| SEE ALSO | intLib , ioLib , <i>VxWorks Programmer's Guide: I/O System</i> , <i>windsh</i> , <i>Tornado User's Guide: Shell</i> |

ipFilterLib

| | |
|--------------------|--|
| NAME | ipFilterLib – ip filter hooks library |
| ROUTINES | <i>ipFilterLibInit()</i> – initialize ip filter facility <i>ipFilterHookAdd()</i> – add a routine to receive all internet protocol packets <i>ipFilterHookDelete()</i> – delete a ip filter hook routine |
| DESCRIPTION | This library provides utilities that give direct access to IP packets. Incoming raw IP packets can be examined or processed using the hooks <i>ipFilterHookAdd()</i> . The input hook can be used to receive raw IP packets that are a part of IP (Internet Protocol) protocols. The filter hook can also be used to build IP traffic monitoring and testing tools. Normally, the network should be accessed through the higher-level socket interface provided in sockLib . The routines in ipFilterLib should rarely, if ever, be necessary for applications. The <i>ipFilterLibInit()</i> routine links the ip filtering facility into the VxWorks system. This is performed automatically if <code>INCLUDE_IP_FILTER</code> is defined in <code>configAll.h</code> . |
| SEE ALSO | <i>VxWorks Programmer's Guide: Network</i> |

ipProto

| | |
|----------------------|---|
| NAME | ipProto – an interface between the BSD IP protocol and the MUX |
| ROUTINES | <i>ipAttach()</i> – a generic attach routine for the TCP/IP network stack <i>ipDetach()</i> – a generic detach routine for the TCP/IP network stack |
| DESCRIPTION | This library provides an interface between the Berkeley protocol stack and the MUX interface. The <i>ipAttach()</i> routine binds the IP protocol to a specific device. It is called automatically during network initialization if <code>INCLUDE_END</code> is defined. The <i>ipDetach()</i> routine removes an existing binding. |
| INCLUDE FILES | end.h muxLib.h etherMultiLib.h sys/ioctl.h etherLib.h |

kernelLib

| | |
|--------------------|--|
| NAME | kernelLib – VxWorks kernel library |
| ROUTINES | <i>kernelInit()</i> – initialize the kernel <i>kernelVersion()</i> – return the kernel revision string <i>kernelTimeSlice()</i> – enable round-robin selection |
| DESCRIPTION | The VxWorks kernel provides tasking control services to an application. The libraries kernelLib , taskLib , semLib , tickLib , and wdLib comprise the kernel functionality. This library is the interface to the VxWorks kernel initialization, revision information, and scheduling control. |

KERNEL INITIALIZATION

The kernel must be initialized before any other kernel operation is performed. Normally kernel initialization is taken care of by the system configuration code in *usrInit()* in **usrConfig.c**.

Kernel initialization consists of the following:

- (1) Defining the starting address and size of the system memory partition. The *malloc()* routine uses this partition to satisfy memory allocation requests of other facilities in VxWorks.
- (2) Allocating the specified memory size for an interrupt stack. Interrupt service routines will use this stack unless the underlying architecture does not support a separate interrupt stack, in which case the service routine will use the stack of the interrupted task.

- (3) Specifying the interrupt lock-out level. VxWorks will not exceed the specified level during any operation. The lock-out level is normally defined to mask the highest priority possible. However, in situations where extremely low interrupt latency is required, the lock-out level may be set to ensure timely response to the interrupt in question. Interrupt service routines handling interrupts of priority greater than the interrupt lock-out level may not call any VxWorks routine.

Once the kernel initialization is complete, a root task is spawned with the specified entry point and stack size. The root entry point is normally *usrRoot()* of the *usrConfig.c* module. The remaining VxWorks initialization takes place in *usrRoot()*.

ROUND-ROBIN SCHEDULING

Round-robin scheduling allows the processor to be shared fairly by all tasks of the same priority. Without round-robin scheduling, when multiple tasks of equal priority must share the processor, a single non-blocking task can usurp the processor until preempted by a task of higher priority, thus never giving the other equal-priority tasks a chance to run.

Round-robin scheduling is disabled by default. It can be enabled or disabled with the routine *kernelTimeSlice()*, which takes a parameter for the "time slice" (or interval) that each task will be allowed to run before relinquishing the processor to another equal-priority task. If the parameter is zero, round-robin scheduling is turned off. If round-robin scheduling is enabled and preemption is enabled for the executing task, the routine *tickAnnounce()* will increment the task's time-slice count. When the specified time-slice interval is completed, the counter is cleared and the task is placed at the tail of the list of tasks at its priority. New tasks joining a given priority group are placed at the tail of the group with a run-time counter initialized to zero.

If a higher priority task preempts a task during its time-slice, the time-slice of the preempted task count is not changed for the duration of the preemption. If preemption is disabled during round-robin scheduling, the time-slice count of the executing task is not incremented.

INCLUDE FILES **kernelLib.h**

SEE ALSO **taskLib, intLib**, *VxWorks Programmer's Guide: Basic OS*

ledLib

NAME ledLib – line-editing library

ROUTINES *ledOpen()* – create a new line-editor ID
ledClose() – discard the line-editor ID
ledRead() – read a line with line-editing
ledControl() – change the line-editor ID parameters

DESCRIPTION This library provides a line-editing layer on top of a **tty** device. The shell uses this interface for its history-editing features.

The shell history mechanism is similar to the UNIX Korn shell history facility, with a built-in line-editor similar to UNIX **vi** that allows previously typed commands to be edited. The command *h()* displays the 20 most recent commands typed into the shell; old commands fall off the top as new ones are entered.

To edit a command, type ESC to enter edit mode, and use the commands listed below. The ESC key switches the shell to edit mode. The RETURN key always gives the line to the shell from either editing or input mode.

The following list is a summary of the commands available in edit mode.

Movement and search commands:

nG – Go to command number *n*.
/s – Search for string *s* backward in history.
?s – Search for string *s* forward in history.
n – Repeat last search.
N – Repeat last search in opposite direction.
nk – Get *n*th previous shell command in history.
n- – Same as *k*.
nj – Get *n*th next shell command in history.
n+ – Same as *j*.
nh – Move left *n* characters.
CTRL+H – Same as *h*.
nl – (letter *el*) Move right *n* characters.
SPACE – Same as *l*.
nw – Move *n* words forward.
nW – Move *n* blank-separated words forward.
ne – Move to end of the *n*th next word.
nE – Move to end of the *n*th next blank-separated word.
nb – Move back *n* words.
nB – Move back *n* blank-separated words.

- f***c* – Find character *c*, searching forward.
- F***c* – Find character *c*, searching backward.
- ^** – Move cursor to first non-blank character in line.
- \$** – Go to end of line.
- 0** – Go to beginning of line.

Insert commands (input is expected until an ESC is typed):

- a** – Append.
- A** – Append at end of line.
- c** **SPACE** – Change character.
- cl** – Change character.
- cw** – Change word.
- cc** – Change entire line.
- c\$** – Change everything from cursor to end of line.
- C** – Same as **c\$**.
- S** – Same as **cc**.
- i** – Insert.
- I** – Insert at beginning of line.
- R** – Type over characters.

Editing commands:

- nrc** – Replace the following *n* characters with *c*.
- nx** – Delete *n* characters starting at cursor.
- nX** – Delete *n* characters to the left of the cursor.
- d** **SPACE** – Delete character.
- dl** – Delete character.
- dw** – Delete word.
- dd** – Delete entire line.
- d\$** – Delete everything from cursor to end of line.
- D** – Same as **d\$**.
- p** – Put last deletion after the cursor.
- P** – Put last deletion before the cursor.
- u** – Undo last command.
- ~** – Toggle case, lower to upper or vice versa.

Special commands:

- CTRL+U** – Delete line and leave edit mode.
- CTRL+L** – Redraw line.
- CTRL+D** – Complete symbol name.
- RETURN** – Give line to shell and leave edit mode.

The default value for *n* is 1.

| | |
|----------------------|--|
| DEFICIENCIES | Since the shell toggles between raw mode and line mode, type-ahead can be lost. The ESC, redraw, and non-printable characters are built-in. The EOF, backspace, and line-delete are not imported well from tyLib . Instead, tyLib should supply and/or support these characters via <i>ioctl()</i> . Some commands do not take counts as users might expect. For example, "mi" will not insert whatever was entered <i>n</i> times. |
| INCLUDE FILES | ledLib.h |
| SEE ALSO | <i>VxWorks Programmer's Guide: Shell</i> |

In97xEnd

| | |
|--------------------|--|
| NAME | In97xEnd – END style AMD Am79C97X PCnet-PCI Ethernet driver |
| ROUTINES | <i>In97xEndLoad()</i> – initialize the driver and device <i>In97xInitParse()</i> – parse the initialization string |
| DESCRIPTION | <p>This module implements the Advanced Micro Devices Am79C971 Am79C972 and Am79C973 PCnet-PCI Ethernet 32 bit network interface driver.</p> <p>The PCnet-PCI ethernet controller is inherently little endian because the chip is designed to operate on a PCI bus which is a little endian bus. The software interface to the driver is divided into three parts. The first part is the PCI configuration registers and their set up. This part is done at the BSP level in the various BSPs which use this driver. The second and third part are dealt in the driver. The second part of the interface comprises of the I/O control registers and their programming. The third part of the interface comprises of the descriptors and the buffers.</p> <p>This driver is designed to be moderately generic, operating unmodified across the range of architectures and targets supported by VxWorks. To achieve this, the driver must be given several target-specific parameters, and some external support routines must be provided. These target-specific values and the external support routines are described below.</p> <p>This driver supports multiple units per CPU. The driver can be configured to support big-endian or little-endian architectures. It contains error recovery code to handle known device errata related to DMA activity.</p> <p>Big endian processors can be connected to the PCI bus through some controllers which take care of hardware byte swapping. In such cases all the registers which the chip DMA s to have to be swapped and written to, so that when the hardware swaps the accesses, the chip would see them correctly. The chip still has to be programmed to operated in little endian mode as it is on the PCI bus. If the cpu board hardware automatically swaps all the</p> |

accesses to and from the PCI bus, then input and output byte stream need not be swapped.

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

The only external interface is the *In97xEndLoad()* routine, which expects the *initString* parameter as input. This parameter passes in a colon-delimited string of the format:

```
unit:devMemAddr:devIoAddr:pciMemBase:<vecNum:intLvl:memAdrs:  
memSize:memWidth:csr3b:offset:flags
```

The *In97xEndLoad()* function uses *strtok()* to parse the string.

TARGET-SPECIFIC PARAMETERS

unit

A convenient holdover from the former model. This parameter is used only in the string name for the driver.

devMemAddr

This parameter in the memory base address of the device registers in the memory map of the CPU. It indicates to the driver where to find the RDP register. The LANCE presents two registers to the external interface, the RDP (register data port) and RAP (register address port) registers. This driver assumes that these two registers occupy two unique addresses in a memory space that is directly accessible by the CPU executing this driver. The driver assumes that the RDP register is mapped at a lower address than the RAP register; the RDP register is therefore derived from the "base address." This parameter should be equal to NONE if memory map is not used.

devIoAddr

This parameter in the IO base address of the device registers in the IO map of some CPUs. It indicates to the driver where to find the RDP register. If both *devIoAddr* and *devMemAddr* are given then the device chooses *devMemAddr* which is a memory mapped register base address. This parameter should be equal to NONE if IO map is not used.

pciMemBase

This parameter is the base address of the CPU memory as seen from the PCI bus. This parameter is zero for most intel architectures.

vecNum

This parameter is the vector associated with the device interrupt. This driver configures the LANCE device to generate hardware interrupts for various events within the device; thus it contains an interrupt handler routine. The driver calls *intConnect()* to connect its interrupt handler to the interrupt vector generated as a result of the LANCE interrupt.

intLvl

Some targets use additional interrupt controller devices to help organize and service the various interrupt sources. This driver avoids all board-specific knowledge of such devices. During the driver's initialization, the external routine **sysLan97xIntEnable()** is called to perform any board-specific operations required to allow the servicing of a LANCE interrupt. For a description of **sysLan97xIntEnable()**, see "External Support Requirements" below.

memAdrs

This parameter gives the driver the memory address to carve out its buffers and data structures. If this parameter is specified to be NONE then the driver allocates cache coherent memory for buffers and descriptors from the system pool. The LANCE device is a DMA type of device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the LANCE. It assumes that this shared memory is directly available to it without any arbitration or timing concerns.

memSize

This parameter can be used to explicitly limit the amount of shared memory (bytes) this driver will use. The constant NONE can be used to indicate no specific size limitation. This parameter is used only if a specific memory region is provided to the driver.

memWidth

Some target hardware that restricts the shared memory region to a specific location also restricts the access width to this region by the CPU. On these targets, performing an access of an invalid width will cause a bus error.

This parameter can be used to specify the number of bytes of access width to be used by the driver during access to the shared memory. The constant NONE can be used to indicate no restrictions.

Current internal support for this mechanism is not robust; implementation may not work on all targets requiring these restrictions.

csr3b

The LANCE control register #3 determines the bus mode of the device, allowing the support of big-endian and little-endian architectures. This parameter, defined as "UINT32 InCSR_3B", is the value that will be placed into LANCE control register #3. The default value supports Motorola-type buses. For information about changing this parameter, see the manual. Normally for devices on the PCI bus this should always be little endian. This value is zero normally

offset

This parameter specifies the offset from which the packet has to be loaded from the beginning of the device buffer. Normally this parameter is zero except for architectures which access long words only on aligned addresses. For these architectures the value of this offset should be 2.

flags

This parameter is used for future use, currently its value should be zero.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires several external support functions, defined as macros:

```

SYS_INT_CONNECT(pDrvCtrl, routine, arg)
SYS_INT_DISCONNECT (pDrvCtrl, routine, arg)
SYS_INT_ENABLE(pDrvCtrl)
SYS_INT_DISABLE(pDrvCtrl)
SYS_OUT_BYTE(pDrvCtrl, reg, data)
SYS_IN_BYTE(pDrvCtrl, reg, data)
SYS_OUT_WORD(pDrvCtrl, reg, data)
SYS_IN_WORD(pDrvCtrl, reg, data)
SYS_OUT_LONG(pDrvCtrl, reg, data)
SYS_IN_LONG(pDrvCtrl, reg, data)
SYS_ENET_ADDR_GET(pDrvCtrl, pAddress)
sysLan97xIntEnable(pDrvCtrl->intLevel)
sysLan97xIntDisable(pDrvCtrl->intLevel)
sysLan97xEnetAddrGet(pDrvCtrl, enetAdrs)

```

There are default values in the source code for these macros. They presume memory mapped accesses to the device registers and the normal *intConnect()*, and *intEnable()* BSP functions. The first argument to each is the device controller structure. Thus, each has access back to all the device-specific information. Having the pointer in the macro facilitates the addition of new features to this driver.

The macros *SYS_INT_CONNECT*, *SYS_INT_DISCONNECT*, *SYS_INT_ENABLE*, and *SYS_INT_DISABLE* allow the driver to be customized for BSPs that use special versions of these routines.

The macro *SYS_INT_CONNECT* is used to connect the interrupt handler to the appropriate vector. By default it is the routine *intConnect()*.

The macro *SYS_INT_DISCONNECT* is used to disconnect the interrupt handler prior to unloading the module. By default this is a dummy routine that returns OK.

The macro *SYS_INT_ENABLE* is used to enable the interrupt level for the end device. It is called once during initialization. It calls an external board level routine *sysLan97xIntEnable()*.

The macro *SYS_INT_DISABLE* is used to disable the interrupt level for the end device. It is called during stop. It calls an external board level routine *sysLan97xIntDisable()*.

The macro *SYS_ENET_ADDR_GET* is used to get the ethernet hardware of the chip. This macro calls an external board level routine namely *sysLan97xEnetAddrGet()* to get the ethernet address.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- 13288 bytes in text for a I80486 target
- 64 bytes in the initialized data section (data)
- 0 bytes in the uninitialized data section (BSS)

The driver allocates clusters of size 1520 bytes for receive frames and and transmit frames.

INCLUDES `end.h endLib.h etherMultiLib.h ln97xEnd.h`

SEE ALSO `muxLib`, `endLib`, `netBufLib` *Writing and Enhanced Network Driver Advanced Micro Devices PCnet-PCI Ethernet Controller for PCI.*

In7990End

NAME `In7990End` – END style AMD 7990 LANCE Ethernet network interface driver

ROUTINES `ln7990EndLoad()` – initialize the driver and device
`ln7990InitParse()` – parse the initialization string
`ln7990InitMem()` – initialize memory for Lance chip

DESCRIPTION This module implements the Advanced Micro Devices Am7990 LANCE Ethernet network interface driver. The driver can be configured to support big-endian or little-endian architectures, and it contains error recovery code to handle known device errata related to DMA activity.

This driver is designed to be moderately generic. Thus, it operates unmodified across the range of architectures and targets supported by VxWorks. To achieve this, the driver load routine requires an input string consisting of several target-specific values. The driver also requires some external support routines. These target-specific values and the external support routines are described below. If any of the assumptions stated below are not true for your particular hardware, this driver might not function correctly with that hardware.

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

The only external interface is the `ln7990EndLoad()` routine, which expects the `initString` parameter as input. This parameter passes in a colon-delimited string of the format:

```
unit:CSR_reg_addr:RAP_reg_addr:int_vector:int_level:shmem_addr:shmem_size:shmem_width:of  
fset:csr3B
```

The *In7990EndLoad()* function uses *strtok()* to parse the string.

TARGET-SPECIFIC PARAMETERS

unit

A convenient holdover from the former model. This parameter is used only in the string name for the driver.

CSR_register_addr

Tells the driver where to find the CSR register.

RAP_register_addr

Tells the driver where to find the RAP register.

int_vector

Configures the LANCE device to generate hardware interrupts for various events within the device. Thus, it contains an interrupt handler routine. The driver calls *sysIntConnect()* to connect its interrupt handler to the interrupt vector generated as a result of the LANCE interrupt.

int_level

This parameter is passed to an external support routine, *sysLanIntEnable()*, which is described below in "External Support Requirements." This routine is called during as part of driver's initialization. It handles any board-specific operations required to allow the servicing of a LANCE interrupt on targets that use additional interrupt controller devices to help organize and service the various interrupt sources. This parameter makes it possible for this driver to avoid all board-specific knowledge of such devices.

shmem_addr

The LANCE device is a DMA type of device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the LANCE. It assumes that this shared memory is directly available to it without any arbitration or timing concerns.

This parameter can be used to specify an explicit memory region for use by the LANCE. This should be done on hardware that restricts the LANCE to a particular memory region. The constant NONE can be used to indicate that there are no memory limitations, in which case, the driver attempts to allocate the shared memory from the system space.

shmem_size

Use this parameter to explicitly limit the amount of shared memory (bytes) that this driver uses. Use "NONE" to indicate that there is no specific size limitation. This parameter is used only if a specific memory region is provided to the driver.

shmem_width

Some target hardware that restricts the shared memory region to a specific location also restricts the access width to this region by the CPU. On such targets, performing an access of an invalid width causes a bus error. Use this parameter to specify the

number of bytes on which data must be aligned if it is to be used by the driver during access to the shared memory. Use "NONE" to indicate that there are no restrictions. The support for this mechanism is not robust. Thus, its current implementation might not work on all targets requiring these restrictions.

offset

Specifies the memory alignment offset.

csr3B

Specifies the value that is placed into LANCE control register #3. This value determines the bus mode of the device and thus allows the support of big-endian and little-endian architectures. The default value supports Motorola-type buses. Normally this value is 0x4. For SPARC CPUs, it is normally set to 0x7 to add the ACON and BCON control bits. For more information on this register and the bus mode of the LANCE controller, see *Advanced Micro Devices Local Area Network Controller Am7990 (LANCE)*.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires several external support functions, defined as macros:

```
SYS_INT_CONNECT(pDrvCtrl, routine, arg)
SYS_INT_DISCONNECT (pDrvCtrl, routine, arg)
SYS_INT_ENABLE(pDrvCtrl)
SYS_OUT_SHORT(pDrvCtrl, reg, data)
SYS_IN_SHORT(pDrvCtrl, reg, pData)
```

There are default values in the source code for these macros. They presume memory-mapped accesses to the device registers and the normal *intConnect()*, and *intEnable()* BSP functions. The first argument to each is the device controller structure. Thus, each has access back to all the device-specific information. Having the pointer in the macro facilitates the addition of new features to this driver.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one interrupt vector
- 68 bytes in the initialized data section (data)
- 0 bytes of bss

The above data and BSS requirements are for the MC68020 architecture and can vary for other architectures. Code size (text) varies greatly between architectures and is therefore not quoted here.

If the driver is not given a specific region of memory using the *In7990EndLoad()* routine, then it calls *cacheDmaMalloc()* to allocate the memory to be shared with the LANCE. The size requested is 80,542 bytes. If a memory region is provided to the driver, the size of this region is adjustable to suit user needs.

The LANCE can only be operated if the shared memory region is write-coherent with the data cache. The driver cannot maintain cache coherency for data that is written by the driver. That is because members within the shared structures are asynchronously modified by both the driver and the device, and these members might share the same cache line.

SEE ALSO *muxLib, Advanced Micro Devices Local Area Network Controller Am7990 (LANCE)*

loadLib

NAME *loadLib* – object module loader

ROUTINES *loadModule()* – load an object module into memory
loadModuleAt() – load an object module into memory

DESCRIPTION This library provides a generic object module loading facility. Any supported format files may be loaded into memory, relocated properly, their external references resolved, and their external definitions added to the system symbol table for use by other modules and from the shell. Modules may be loaded from any I/O stream which allows repositioning of the pointer. This includes **netDrv**, **nfs**, or local file devices. It does not include sockets.

EXAMPLE

```
fdX = open ("/devX/objFile", O_RDONLY);
loadModule (fdX, LOAD_ALL_SYMBOLS);
close (fdX);
```

This code fragment would load the object file "objFile" located on device "/devX/" into memory which would be allocated from the system memory pool. All external and static definitions from the file would be added to the system symbol table.

This could also have been accomplished from the shell, by typing:

```
-> ld (1) </devX/objFile
```

INCLUDE FILE *loadLib.h*

SEE ALSO *usrLib, symLib, memLib, VxWorks Programmer's Guide: Basic OS*

loginLib

| | |
|---------------------|--|
| NAME | loginLib – user login/password subroutine library |
| ROUTINES | <i>loginInit()</i> – initialize the login table <i>loginUserAdd()</i> – add a user to the login table <i>loginUserDelete()</i> – delete a user entry from the login table <i>loginUserVerify()</i> – verify a user name and password in the login table <i>loginUserShow()</i> – display the user login table <i>loginPrompt()</i> – display a login prompt and validate a user entry <i>loginStringSet()</i> – change the login string <i>loginEncryptInstall()</i> – install an encryption routine <i>loginDefaultEncrypt()</i> – default password encryption routine |
| DESCRIPTION | <p>This library provides a login/password facility for network access to the VxWorks shell. When installed, it requires a user name and password match to gain access to the VxWorks shell from rlogin or telnet. Therefore VxWorks can be used in secure environments where access must be restricted.</p> <p>Routines are provided to prompt for the user name and password, and verify the response by looking up the name/password pair in a login user table. This table contains a list of user names and encrypted passwords that will be allowed to log in to the VxWorks shell remotely. Routines are provided to add, delete, and access the login user table. The list of user names can be displayed with <i>loginUserShow()</i>.</p> |
| INSTALLATION | <p>The login security feature is initialized by the root task, <i>usrRoot()</i>, in <i>usrConfig.c</i>, if the configuration macro INCLUDE_SECURITY is defined. Defining this macro also adds a single default user to the login table. The default user and password are defined as LOGIN_USER_NAME and LOGIN_PASSWORD. These can be set to any desired name and password. More users can be added by making additional calls to <i>loginUserAdd()</i>. If INCLUDE_SECURITY is not defined, access to VxWorks will not be restricted and secure.</p> <p>The name/password pairs are added to the table by calling <i>loginUserAdd()</i>, which takes the name and an encrypted password as arguments. The VxWorks host tool <i>vxencrypt</i> is used to generate the encrypted form of a password. For example, to add a user name of "fred" and password of "flintstone", first run <i>vxencrypt</i> on the host to find the encryption of "flintstone" as follows:</p> <pre>% vxencrypt please enter password: flintstone encrypted password is ScebRezb9c</pre> <p>Then invoke the routine <i>loginUserAdd()</i> in VxWorks:</p> <pre>loginUserAdd ("fred", "ScebRezb9c");</pre> <p>This can be done from the shell, a start-up script, or application code.</p> |

LOGGING IN When the login security facility is installed, every attempt to rlogin or telnet to the VxWorks shell will first prompt for a user name and password.

```
% rlogin target
VxWorks login: fred
Password: flintstone
->
```

The delay in prompting between unsuccessful logins is increased linearly with the number of attempts, in order to slow down password-guessing programs.

ENCRYPTION ALGORITHM

This library provides a simple default encryption routine, *loginDefaultEncrypt()*. This algorithm requires that passwords be at least 8 characters and no more than 40 characters.

The routine *loginEncryptInstall()* allows a user-specified encryption function to be used instead of the default.

INCLUDE FILES **loginLib.h**

SEE ALSO **shellLib**, **vxencrypt**, *VxWorks Programmer's Guide: Shell*

logLib

NAME **logLib** – message logging library

ROUTINES *logInit()* – initialize message logging library
logMsg() – log a formatted error message
logFdSet() – set the primary logging file descriptor
logFdAdd() – add a logging file descriptor
logFdDelete() – delete a logging file descriptor
logTask() – message-logging support task

DESCRIPTION This library handles message logging. It is usually used to display error messages on the system console, but such messages can also be sent to a disk file or printer.

The routines *logMsg()* and *logTask()* are the basic components of the logging system. The *logMsg()* routine has the same calling sequence as *printf()*, but instead of formatting and outputting the message directly, it sends the format string and arguments to a message queue. The task *logTask()* waits for messages on this message queue. It formats each message according to the format string and arguments in the message, prepends the ID of the sender, and writes it on one or more file descriptors that have been specified as logging output streams (by *logInit()* or subsequently set by *logFdSet()* or *logFdAdd()*).

USE IN INTERRUPT SERVICE ROUTINES

Because *logMsg()* does not directly cause output to I/O devices, but instead simply writes to a message queue, it can be called from an interrupt service routine as well as from tasks. Normal I/O, such as *printf()* output to a serial port, cannot be done from an interrupt service routine.

DEFERRED LOGGING

Print formatting is performed within the context of *logTask()*, rather than the context of the task calling *logMsg()*. Since formatting can require considerable stack space, this can reduce stack sizes for tasks that only need to do I/O for error output.

However, this also means that the arguments to *logMsg()* are not interpreted at the time of the call to *logMsg()*, but rather are interpreted at some later time by *logTask()*. This means that the arguments to *logMsg()* should not be pointers to volatile entities. For example, pointers to dynamic or changing strings and buffers should not be passed as arguments to be formatted. Thus the following would not give the desired results:

```
doLog (which)
{
    char string [100];
    strcpy (string, which ? "hello" : "goodbye");
    ...
    logMsg (string);
}
```

By the time *logTask()* formats the message, the stack frame of the caller may no longer exist and the pointer *string* may no longer be valid. On the other hand, the following is correct since the string pointer passed to the *logTask()* always points to a static string:

```
doLog (which)
{
    char *string;
    string = which ? "hello" : "goodbye";
    ...
    logMsg (string);
}
```

INITIALIZATION To initialize the message logging facilities, the routine *logInit()* must be called before calling any other routine in this module. This is done by the root task, *usrRoot()*, in *usrConfig.c*.

INCLUDE FILES *logLib.h*

SEE ALSO *msgQLib*, *VxWorks Programmer's Guide: I/O System*

lptDrv

| | |
|--------------------|--|
| NAME | lptDrv – parallel chip device driver for the IBM-PC LPT |
| ROUTINES | <i>lptDrv()</i> – initialize the LPT driver <i>lptDevCreate()</i> – create a device for an LPT port <i>lptShow()</i> – show LPT statistics |
| DESCRIPTION | This is the driver for the LPT used on the IBM-PC. If <code>INCLUDE_LPT</code> is defined, the driver initializes the LPT on the PC. |

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. However, two routines must be called directly: *lptDrv()* to initialize the driver, and *lptDevCreate()* to create devices.

There are one other callable routines: *lptShow()* to show statistics. The argument to *lptShow()* is a channel number, 0 to 2.

Before the driver can be used, it must be initialized by calling *lptDrv()*. This routine should be called exactly once, before any reads, writes, or calls to *lptDevCreate()*. Normally, it is called from *usrRoot()* in *usrConfig.c*. The first argument to *lptDrv()* is a number of channels, 0 to 2. The second argument is a pointer to the resource table. Definitions of members of the resource table structure are:

```
int  ioBase;          /* IO base address */
int  intVector;      /* interrupt vector */
int  intLevel;       /* interrupt level */
BOOL autofeed;      /* TRUE if enable autofeed */
int  busyWait;       /* loop count for BUSY wait */
int  strobeWait;     /* loop count for STROBE wait */
int  retryCnt;       /* retry count */
int  timeout;        /* timeout second for syncSem */
```

IOCTL FUNCTIONS This driver responds to two functions: `LPT_SETCONTROL` and `LPT_GETSTATUS`. The argument for `LPT_SETCONTROL` is a value of the control register. The argument for `LPT_GETSTATUS` is a integer pointer where a value of the status register is stored.

SEE ALSO *VxWorks Programmer's Guide: I/O System*

IstLib

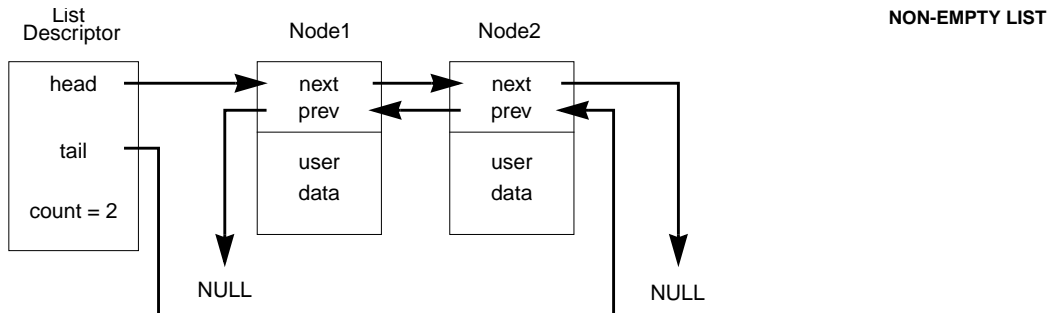
NAME IstLib – doubly linked list subroutine library

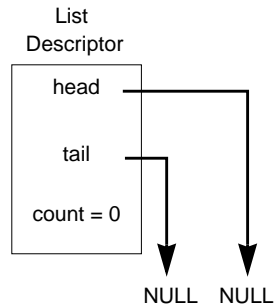
ROUTINES

- IstInit()* – initialize a list descriptor
- IstAdd()* – add a node to the end of a list
- IstConcat()* – concatenate two lists
- IstCount()* – report the number of nodes in a list
- IstDelete()* – delete a specified node from a list
- IstExtract()* – extract a sublist from a list
- IstFirst()* – find first node in list
- IstGet()* – delete and return the first node from a list
- IstInsert()* – insert a node in a list after a specified node
- IstLast()* – find the last node in a list
- IstNext()* – find the next node in a list
- IstNth()* – find the Nth node in a list
- IstPrevious()* – find the previous node in a list
- IstNStep()* – find a list node *nStep* steps away from a specified node
- IstFind()* – find a node in a list
- IstFree()* – free up a list

DESCRIPTION This subroutine library supports the creation and maintenance of a doubly linked list. The user supplies a list descriptor (type LIST) that will contain pointers to the first and last nodes in the list, and a count of the number of nodes in the list. The nodes in the list can be any user-defined structure, but they must reserve space for two pointers as their first elements. Both the forward and backward chains are terminated with a NULL pointer.

The linked-list library simply manipulates the linked-list data structures; no kernel functions are invoked. In particular, linked lists by themselves provide no task synchronization or mutual exclusion. If multiple tasks will access a single linked list, that list must be guarded with some mutual-exclusion mechanism (e.g., a mutual-exclusion semaphore).



EMPTY LIST**INCLUDE FILES** **lstLib.h**

m2IcmpLib

NAME **m2IcmpLib** – MIB-II ICMP-group API for SNMP Agents

ROUTINES *m2IcmpInit()* – initialize MIB-II ICMP-group access
m2IcmpGroupInfoGet() – get the MIB-II ICMP-group global variables
m2IcmpDelete() – delete all resources used to access the ICMP group

DESCRIPTION This library provides MIB-II services for the ICMP group. It provides routines to initialize the group, and to access the group scalar variables. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

USING THIS LIBRARY

This library can be initialized and deleted by calling the routines *m2IcmpInit()* and *m2IcmpDelete()* respectively, if only the ICMP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling *m2Init()* and *m2Delete()*.

The group scalar variables are accessed by calling *m2IcmpGroupInfoGet()* as follows:

```

M2_ICMP   icmpVars;
if (m2IcmpGroupInfoGet (&icmpVars) == OK)
    /* values in icmpVars are valid */
  
```

INCLUDE FILES **m2Lib.h****SEE ALSO** **m2IcmpLib, m2Lib, m2IfLib, m2IpLib, m2TcpLib, m2SysLib**

m2IfLib

| | |
|--------------------|---|
| NAME | m2IfLib – MIB-II interface-group API for SNMP agents |
| ROUTINES | <i>m2IfInit()</i> – initialize MIB-II interface-group routines <i>m2IfGroupInfoGet()</i> – get the MIB-II interface-group scalar variables <i>m2IfTblEntryGet()</i> – get a MIB-II interface-group table entry <i>m2IfTblEntrySet()</i> – set the state of a MIB-II interface entry to UP or DOWN <i>m2IfDelete()</i> – delete all resources used to access the interface group |
| DESCRIPTION | This library provides MIB-II services for the interface group. It provides routines to initialize the group, access the group scalar variables, read the table interfaces and change the state of the interfaces. For a broader description of MIB-II services, see the manual entry for m2Lib . |

USING THIS LIBRARY

This library can be initialized and deleted by calling *m2IfInit()* and *m2IfDelete()* respectively, if only the interface group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling *m2Init()* and *m2Delete()*.

The interface group supports the Simple Network Management Protocol (SNMP) concept of traps, as specified by RFC 1215. The traps supported by this group are "link up" and "link down." This library enables an application to register a hook routine and an argument. This hook routine can be called by the library when a "link up" or "link down" condition is detected. The hook routine must have the following prototype:

```
void TrapGenerator (int trapType, /* M2_LINK_DOWN_TRAP or M2_LINK_UP_TRAP */
                   int interfaceIndex,
                   void * myPrivateArg);
```

The trap routine and argument can be specified at initialization time as input parameters to the routine *m2IfInit()* or to the routine *m2Init()*.

The interface-group global variables can be accessed as follows:

```
M2_INTERFACE ifVars;
if (m2IfGroupInfoGet (&ifVars) == OK)
    /* values in ifVars are valid */
```

An interface table entry can be retrieved as follows:

```
M2_INTERFACETBL interfaceEntry;
/* Specify zero as the index to get the first entry in the table */
interfaceEntry.ifIndex = 2; /* Get interface with index 2 */
if (m2IfTblEntryGet (M2_EXACT_VALUE, &interfaceEntry) == OK)
    /* values in interfaceEntry are valid */
```


An interface entry operational state can be changed as follows:

```
M2_INTERFACETBL ifEntryToSet;
ifEntryToSet.ifIndex      = 2; /* Select interface with index 2      */
                           /* MIB-II value to set the interface */
                           /* to the down state.                  */
ifEntryToSet.ifAdminStatus = M2_ifAdminStatus_down;
if (m2IfTblEntrySet (&ifEntryToSet) == OK)
    /* Interface is now in the down state */
```

INCLUDE FILES m2Lib.h

SEE ALSO m2Lib, m2SysLib, m2IpLib, m2IcmpLib, m2UdpLib, m2TcpLib

m2IpLib

NAME m2IpLib – MIB-II IP-group API for SNMP agents

ROUTINES *m2IpInit()* – initialize MIB-II IP-group access
m2IpGroupInfoGet() – get the MIB-II IP-group scalar variables
m2IpGroupInfoSet() – set MIB-II IP-group variables to new values
m2IpAddrTblEntryGet() – get an IP MIB-II address entry
m2IpAtransTblEntryGet() – get a MIB-II ARP table entry
m2IpAtransTblEntrySet() – add, modify, or delete a MIB-II ARP entry
m2IpRouteTblEntryGet() – get a MIB-2 routing table entry
m2IpRouteTblEntrySet() – set a MIB-II routing table entry
m2IpDelete() – delete all resources used to access the IP group

DESCRIPTION This library provides MIB-II services for the IP group. It provides routines to initialize the group, access the group scalar variables, read the table IP address, route and ARP table. The route and ARP table can also be modified. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

USING THIS LIBRARY

To use this library, the MIB-II interface group must also be initialized; see the manual entry for **m2IfLib**. This library (**m2IpLib**) can be initialized and deleted by calling *m2IpInit()* and *m2IpDelete()* respectively, if only the IP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling *m2Init()* and *m2Delete()*.

The following example demonstrates how to access and change IP scalar variables:

```
M2_IP    ipVars;
int      varToSet;
```

```
if (m2IpGroupInfoGet (&ipVars) == OK)
    /* values in ipVars are valid */
/* if IP is forwarding packets (MIB-II value is 1) turn it off */
if (ipVars.ipForwarding == M2_ipForwarding_forwarding)
    {
    /* Not forwarding (MIB-II value is 2) */
    ipVars.ipForwarding = M2_ipForwarding_not_forwarding;
    varToSet |= M2_IPFORWARDING;
    }
/* change the IP default time to live parameter */
ipVars.ipDefaultTTL = 55;
if (m2IpGroupInfoSet (varToSet, &ipVars) == OK)
    /* values in ipVars are valid */
```

The IP address table is a read-only table. Entries to this table can be retrieved as follows:

```
M2_IPADDR_TBL ipAddrEntry;
/* Specify the index as zero to get the first entry in the table */
ipAddrEntry.ipAdEntAddr = 0; /* Local IP address in host byte order */
/* get the first entry in the table */
if ((m2IpAddrTblEntryGet (M2_NEXT_VALUE, &ipAddrEntry) == OK)
    /* values in ipAddrEntry in the first entry are valid */
/* Process first entry in the table */
/*
 * For the next call, increment the index returned in the previous call.
 * The increment is to the next possible lexicographic entry; for
 * example, if the returned index was 147.11.46.8 the index passed in the
 * next invocation should be 147.11.46.9. If an entry in the table
 * matches the specified index, then that entry is returned.
 * Otherwise the closest entry following it, in lexicographic order,
 * is returned.
 */
/* get the second entry in the table */
if ((m2IpAddrTblEntryGet (M2_NEXT_VALUE, &ipAddrEntryEntry) == OK)
    /* values in ipAddrEntry in the second entry are valid */
```

The IP Address Translation Table (ARP table) includes the functionality of the AT group plus additional functionality. The AT group is supported through this MIB-II table. Entries in this table can be added and deleted. An entry is deleted (with a set operation) by setting the **ipNetToMediaType** field to the MIB-II "invalid" value (2). The following example shows how to delete an entry:

```
M2_IPATRANSTBL atEntry;
/* Specify the index for the connection to be deleted in the table */
atEntry.ipNetToMediaIfIndex = 1 /* interface index */
/* destination IP address in host byte order */
atEntry.ipNetToMediaNetAddress = 0x930b2e08;
/* mark entry as invalid */
```

```

atEntry.ipNetToMediaType      = M2_ipNetToMediaType_invalid;
/* set the entry in the table */
if ((m2IpAtransTblEntrySet (&atEntry) == OK)
    /* Entry deleted successfully */

```

The IP route table allows for entries to be read, deleted, and modified. This example demonstrates how an existing route is deleted:

```

M2_IPROUTETBL      routeEntry;
/* Specify the index for the connection to be deleted in the table */
/* destination IP address in host byte order */
routeEntry.ipRouteDest      = 0x930b2e08;
                                                                    /* mark entry as invalid */
routeEntry.ipRouteType      = M2_ipRouteType_invalid;
/* set the entry in the table */
if ((m2IpRouteTblEntrySet (M2_IP_ROUTE_TYPE, &routeEntry) == OK)
    /* Entry deleted successfully */

```

INCLUDE FILES m2Lib.h

SEE ALSO m2Lib, m2SysLib, m2IfLib, m2IcmpLib, m2UdpLib, m2TcpLib

m2Lib

NAME m2Lib – MIB-II API library for SNMP agents

ROUTINES *m2Init()* – initialize the SNMP MIB-2 library
m2Delete() – delete all the MIB-II library groups

DESCRIPTION This library provides Management Information Base (MIB-II, defined in RFC 1213) services for applications wishing to have access to MIB parameters.

There are no specific provisions for MIB-I: all services are provided at the MIB-II level. Applications that use this library for MIB-I must hide the MIB-II extensions from higher level protocols. The library accesses all the MIB-II parameters, and presents them to the application in data structures based on the MIB-II specifications.

The routines provided by the VxWorks MIB-II library are separated into groups that follow the MIB-II definition. Each supported group has its own interface library:

m2SysLib
 systems group

m2IfLib
 interface group

m2Lib

m2IpLib

IP group (includes AT)

m2IcmpLib

ICMP group

m2TcpLib

TCP group

m2UdpLib

UDP group

MIB-II retains the AT group for backward compatibility, but includes its functionality in the IP group. The EGP and SNMP groups are not supported by this interface. The variables in each group have been subdivided into two types: table entries and scalar variables. Each type has a pair of routines that get and set the variables.

USING THIS LIBRARY

There are four types of operations on each group:

- initializing the group
- getting variables and table entries
- setting variables and table entries
- deleting the group

Only the groups that are to be used need be initialized. There is one exception: to use the IP group, the interface group must also be initialized. Applications that require MIB-II support from all groups can initialize all groups at once by calling the *m2Init()*. All MIB-II group services can be disabled by calling *m2Delete()*. Applications that need access only to a particular set of groups need only call the initialization routines of the desired groups.

To read the scalar variables for each group, call one of the following routines:

```
m2SysGroupInfoGet()  
m2IfGroupInfoGet()  
m2IpGroupInfoGet()  
m2IcmpGroupInfoGet()  
m2TcpGroupInfoGet()  
m2UdpGroupInfoGet()
```

The input parameter to the routine is always a pointer to a structure specific to the associated group. The scalar group structures follow the naming convention "M2_groupname". The get routines fill in the input structure with the values of all the group variables.

The scalar variables can also be set to a user supplied value. Not all groups permit setting variables, as specified by the MIB-II definition. The following group routines allow setting variables:

m2SysGroupInfoSet()
m2IpGroupInfoSet()

The input parameters to the variable-set routines are a bit field that specifies which variables to set, and a group structure. The structure is the same structure type used in the get operation. Applications need set only the structure fields corresponding to the bits that are set in the bit field.

The MIB-II table routines read one entry at a time. Each MIB-II group that has tables has a get routine for each table. The following table-get routines are available:

m2IfTblEntryGet()
m2IpAddrTblEntryGet()
m2IpAtransTblEntryGet()
m2IpRouteTblEntryGet()
m2TcpConnEntryGet()
m2UdpTblEntryGet()

The input parameters are a pointer to a table entry structure, and a flag value specifying one of two types of table search. Each table entry is a structure, where the struct type name follows this naming convention: "M2_GroupnameTablenameTBL". The MIB-II RFC specifies an index that identifies a table entry. Each get request must specify an index value. To retrieve the first entry in a table, set all the index fields of the table-entry structure to zero, and use the search parameter **M2_NEXT_VALUE**. To retrieve subsequent entries, pass the index returned from the previous invocation, incremented to the next possible lexicographical entry. The search field can only be set to the constants **M2_NEXT_VALUE** or **M2_EXACT_VALUE**:

M2_NEXT_VALUE

retrieves a table entry that is either identical to the index value specified as input, or is the closest entry following that value, in lexicographic order.

M2_EXACT_VALUE

retrieves a table entry that exactly matches the index specified in the input structure.

Some MIB-II table entries can be added, modified and deleted. Routines to manipulate such entries are described in the manual pages for individual groups.

All the IP network addresses that are exchanged with the MIB-II library must be in host-byte order; use *ntohl()* to convert addresses before calling these library routines.

The following example shows how to initialize the MIB-II library for all groups.

```
extern FUNCPTR myTrapGenerator;
extern void * myTrapGeneratorArg;
M2_OBJECTID mySysObjectId = { 8, {1,3,6,1,4,1,731,1} };
if (m2Init ("VxWorks 5.1.1 MIB-II library (sysDescr)",
           "support@wrs.com (sysContact)",
           "1010 Atlantic Avenue Alameda, California 94501
(sysLocation)",
           &mySysObjectId,
```

```
        myTrapGenerator,  
        myTrapGeneratorArg,  
        0) == OK)  
/* MIB-II groups initialized successfully */
```

INCLUDE FILES **m2Lib.h**

SEE ALSO **m2IfLib, m2IpLib, m2IcmpLib, m2UdpLib, m2TcpLib, m2SysLib**

m2SysLib

NAME **m2SysLib** – MIB-II system-group API for SNMP agents

ROUTINES *m2SysInit()* – initialize MIB-II system-group routines
m2SysGroupInfoGet() – get system-group MIB-II variables
m2SysGroupInfoSet() – set system-group MIB-II variables to new values
m2SysDelete() – delete resources used to access the MIB-II system group

DESCRIPTION This library provides MIB-II services for the system group. It provides routines to initialize the group and to access the group scalar variables. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

USING THIS LIBRARY

This library can be initialized and deleted by calling *m2SysInit()* and *m2SysDelete()* respectively, if only the system group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling *m2Init()* and *m2Delete()*.

The system group provides the option to set the system variables at the time *m2Sysinit()* is called. The MIB-II variables **sysDescr** and **sysobjectId** are read-only, and can be set only by the system-group initialization routine. The variables **sysContact**, **sysName** and **sysLocation** can be set through *m2SysGroupInfoSet()* at any time.

The following is an example of system group initialization:

```
M2_OBJECTID mySysObjectId = { 8, {1,3,6,1,4,1,731,1} };  
if (m2SysInit ("VxWorks MIB-II library ",  
             "support@wrs.com",  
             "1010 Atlantic Avenue Alameda, California 94501",  
             &mySysObjectId) == OK)  
/* System group initialized successfully */
```

The system group variables can be accessed as follows:

```
M2_SYSTEM    sysVars;
```

```
if (m2SysGroupInfoGet (&sysVars) == OK)
    /* values in sysVars are valid */
```

The system group variables can be set as follows:

```
M2_SYSTEM    sysVars;
unsigned int varToSet;    /* bit field of variables to set */
/* Set the new system Name */
strcpy (m2SysVars.sysName, "New System Name");
varToSet |= M2SYSNAME;
/* Set the new contact name */
strcpy (m2SysVars.sysContact, "New Contact");
varToSet |= M2SYSCONTACT;
if (m2SysGroupInfoGet (varToSet, &sysVars) == OK)
    /* values in sysVars set */
```

INCLUDE FILES m2Lib.h

SEE ALSO m2Lib, m2IfLib, m2IpLib, m2IcmpLib, m2UdpLib, m2TcpLib

m2TcpLib

NAME m2TcpLib – MIB-II TCP-group API for SNMP agents

ROUTINES *m2TcpInit()* – initialize MIB-II TCP-group access
m2TcpGroupInfoGet() – get MIB-II TCP-group scalar variables
m2TcpConnEntryGet() – get a MIB-II TCP connection table entry
m2TcpConnEntrySet() – set a TCP connection to the closed state
m2TcpDelete() – delete all resources used to access the TCP group

DESCRIPTION This library provides MIB-II services for the TCP group. It provides routines to initialize the group, access the group global variables, read the table of TCP connections, and change the state of a TCP connection. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

USING THIS LIBRARY

This library can be initialized and deleted by calling *m2TcpInit()* and *m2TcpDelete()* respectively, if only the TCP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling *m2Init()* and *m2Delete()*.

The group global variables are accessed by calling *m2TcpGroupInfoGet()* as follows:

```
M2_TCP    tcpVars;
```

```
if (m2TcpGroupInfoGet (&tcpVars) == OK)
    /* values in tcpVars are valid */
```

The TCP table of connections can be accessed in lexicographical order. The first entry in the table can be accessed by setting the table index to zero. Every other entry thereafter can be accessed by passing to *m2TcpConnTblEntryGet()* the index retrieved in the previous invocation incremented to the next lexicographical value by giving M2_NEXT_VALUE as the search parameter. For example:

```
M2_TCPCONNTBL tcpEntry;
/* Specify a zero index to get the first entry in the table */
tcpEntry.tcpConnLocalAddress = 0; /* Local IP addr in host byte order */
tcpEntry.tcpConnLocalPort    = 0; /* Local TCP port                */
tcpEntry.tcpConnRemAddress   = 0; /* remote IP address              */
tcpEntry.tcpConnRemPort     = 0; /* remote TCP port in host byte order */
/* get the first entry in the table */
if ((m2TcpConnTblEntryGet (M2_NEXT_VALUE, &tcpEntry) == OK)
    /* values in tcpEntry in the first entry are valid */
/* process first entry in the table */
/*
 * For the next call, increment the index returned in the previous call.
 * The increment is to the next possible lexicographic entry; for
 * example, if the returned index was 147.11.46.8.2000.147.11.46.158.1000
 * the index passed in the next invocation should be
 * 147.11.46.8.2000.147.11.46.158.1001. If an entry in the table
 * matches the specified index, then that entry is returned.
 * Otherwise the closest entry following it, in lexicographic order,
 * is returned.
 */
/* get the second entry in the table */
if ((m2TcpConnTblEntryGet (M2_NEXT_VALUE, &tcpEntry) == OK)
    /* values in tcpEntry in the second entry are valid */
```

The TCP table of connections allows only for a connection to be deleted as specified in the MIB-II. For example:

```
M2_TCPCONNTBL tcpEntry;
/* Fill in the index for the connection to be deleted in the table */
/* Local IP address in host byte order, and local port number */
tcpEntry.tcpConnLocalAddress = 0x930b2e08;
tcpEntry.tcpConnLocalPort    = 3000;
/* Remote IP address in host byte order, and remote port number */
tcpEntry.tcpConnRemAddress   = 0x930b2e9e;
tcpEntry.tcpConnRemPort     = 3000;
tcpEntry.tcpConnState       = 12; /* MIB-II state value for delete */
/* set the entry in the table */
if ((m2TcpConnTblEntrySet (&tcpEntry) == OK)
    /* tcpEntry deleted successfully */
```


INCLUDE FILES **m2Lib.h**

SEE ALSO **m2Lib, m2IfLib, m2IpLib, m2IcmpLib, m2UdpLib, m2SysLib**

m2UdpLib

NAME **m2UdpLib** – MIB-II UDP-group API for SNMP agents

ROUTINES **m2UdpInit()** – initialize MIB-II UDP-group access
m2UdpGroupInfoGet() – get MIB-II UDP-group scalar variables
m2UdpTblEntryGet() – get a UDP MIB-II entry from the UDP list of listeners
m2UdpDelete() – delete all resources used to access the UDP group

DESCRIPTION This library provides MIB-II services for the UDP group. It provides routines to initialize the group, access the group scalar variables, and read the table of UDP listeners. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

USING THIS LIBRARY

This library can be initialized and deleted by calling **m2UdpInit()** and **m2UdpDelete()** respectively, if only the UDP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling **m2Init()** and **m2Delete()**.

The group scalar variables are accessed by calling **m2UdpGroupInfoGet()** as follows:

```
M2_UDP   udpVars;
if (m2UdpGroupInfoGet (&udpVars) == OK)
    /* values in udpVars are valid */
```

The UDP table of listeners can be accessed in lexicographical order. The first entry in the table can be accessed by setting the table index to zero in a call to **m2UdpTblEntryGet()**. Every other entry thereafter can be accessed by incrementing the index returned from the previous invocation to the next possible lexicographical index, and repeatedly calling **m2UdpTblEntryGet()** with the **M2_NEXT_VALUE** constant as the search parameter. For example:

```
M2_UDPTBL  udpEntry;
/* Specify zero index to get the first entry in the table */
udpEntry.udpLocalAddress = 0;   /* local IP addr in host byte order */
udpEntry.udpLocalPort   = 0;   /* local port Number */
/* get the first entry in the table */
if ((m2UdpTblEntryGet (M2_NEXT_VALUE, &udpEntry) == OK)
    /* values in udpEntry in the first entry are valid */
    /* process first entry in the table */
```

```
/*
 * For the next call, increment the index returned in the previous call.
 * The increment is to the next possible lexicographic entry; for
 * example, if returned index was 0.0.0.0.3000 the index passed in the
 * next invocation should be 0.0.0.0.3001. If an entry in the table
 * matches the specified index, then that entry is returned.
 * Otherwise the closest entry following it, in lexicographic order,
 * is returned.
 */
/* get the second entry in the table */
if ((m2UdpTblEntryGet (M2_NEXT_VALUE, &udpEntry) == OK)
    /* values in udpEntry in the second entry are valid */
```

INCLUDE FILES m2Lib.h

SEE ALSO m2Lib, m2IfLib, m2IpLib, m2IcmpLib, m2TcpLib, m2SysLib

m68302Sio

NAME m68302Sio – Motorola MC68302 bimodal tty driver

ROUTINES *m68302SioInit()* – initialize a M68302_CP
 m68302SioInit2() – initialize a M68302_CP (part 2)

DESCRIPTION This is the driver for the internal communications processor (CP) of the Motorola MC68302.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. Before the driver can be used, it must be initialized by calling the routines *m68302SioInit()* and *m68302SioInit2()*. Normally, they are called by *sysSerialHwInit()* and *sysSerialHwInit2()* in *sysSerial.c*

This driver uses 408 bytes of buffer space as follows:

- 128 bytes for portA tx buffer
- 128 bytes for portB tx buffer
- 128 bytes for portC tx buffer
- 8 bytes for portA rx buffers (8 buffers, 1 byte each)
- 8 bytes for portB rx buffers (8 buffers, 1 byte each)
- 8 bytes for portC rx buffers (8 buffers, 1 byte each)

The buffer pointer in the **m68302cp** structure points to the buffer area, which is usually specified as **IMP_BASE_ADDR**.

- IOCTL FUNCTIONS** This driver responds to the same *ioctl()* codes as a normal tty driver; for more information, see the manual entry for **tyLib**. The available baud rates are 300, 600, 1200, 2400, 4800, 9600 and 19200.
- INCLUDE FILES** **drv/sio/m68302Sio.h** **sioLib.h**
- SEE ALSO** **ttyDrv**, **tyLib**

m68332Sio

- NAME** **m68332Sio** – Motorola MC68332 tty driver
- ROUTINES** *m68332DevInit()* – initialize the SCC
m68332Int() – handle an SCC interrupt
- DESCRIPTION** This is the driver for the Motorola MC68332 on-chip UART. It has only one serial channel.
- USAGE** A **M68332_CHAN** structure is used to describe the chip. The BSP's *sysHwInit()* routine typically calls *sysSerialHwInit()*, which initializes all the values in the **M68332_CHAN** structure (except the **SIO_DRV_FUNCS**) before calling *m68332DevInit()*. The BSP's *sysHwInit2()* routine typically calls *sysSerialHwInit2()*, which connects the chips interrupt (**m68332Int**) via *intConnect()*.
- INCLUDE FILES** **drv/sio/m68332Sio.h**

m68360Sio

- NAME** **m68360Sio** – Motorola MC68360 SCC UART serial driver
- ROUTINES** *m68360DevInit()* – initialize the SCC
m68360Int() – handle an SCC interrupt
- DESCRIPTION** This is the driver for the SCC's in the internal Communications Processor (CP) of the Motorola MC68360. This driver only supports the SCC's in asynchronous UART mode.

| | |
|----------------------|---|
| USAGE | A m68360_CHAN structure is used to describe the chip. The BSP's <i>sysHwInit()</i> routine typically calls <i>sysSerialHwInit()</i> which initializes all the values in the M68360_CHAN structure (except the SIO_DRV_FUNCS) before calling <i>m68360DevInit()</i> . The BSP's <i>sysHwInit2()</i> routine typically calls <i>sysSerialHwInit2()</i> which connects the chips interrupt (m68360Int) via <i>intConnect()</i> . |
| INCLUDE FILES | drv/sio/m68360Sio.h |

m68562Sio

| | |
|----------------------|--|
| NAME | m68562Sio – MC68562 DUSCC serial driver |
| ROUTINES | <i>m68562HrdInit()</i> – initialize the DUSCC <i>m68562RxTxErrInt()</i> – handle a receiver/transmitter error interrupt <i>m68562RxInt()</i> – handle a receiver interrupt <i>m68562TxInt()</i> – handle a transmitter interrupt |
| DESCRIPTION | This is the driver for the MC68562 DUSCC serial chip. It uses the DUSCC in asynchronous mode only. |
| USAGE | A M68562_QUSART structure is used to describe the chip. This data structure contains M68562_CHAN structures which describe the chip's serial channels. The BSP's <i>sysHwInit()</i> routine typically calls <i>sysSerialHwInit()</i> which initializes all the values in the M68562_QUSART structure (except the SIO_DRV_FUNCS) before calling <i>m68562HrdInit()</i> . The BSP's <i>sysHwInit2()</i> routine typically calls <i>sysSerialHwInit2()</i> which connects the chips interrupts (m68562RxTxErrInt, m68562RxInt, and m68562TxInt) via <i>intConnect()</i> . |
| IOCTL | This driver responds to the same <i>ioctl()</i> codes as a normal serial driver. See the file <i>sioLib.h</i> for more information. |
| INCLUDE FILES | drv/sio/m68562Sio.h |

m68681Sio

| | |
|--------------------|--|
| NAME | m68681Sio – M68681 serial communications driver |
| ROUTINES | <i>m68681DevInit()</i> – initialize a M68681_DUART <i>m68681DevInit2()</i> – initialize a M68681_DUART , part 2 <i>m68681ImrSetClr()</i> – set and clear bits in the DUART interrupt-mask register <i>m68681Imr()</i> – return the current contents of the DUART interrupt-mask register <i>m68681AcrSetClr()</i> – set and clear bits in the DUART auxiliary control register <i>m68681Acr()</i> – return the contents of the DUART auxiliary control register <i>m68681OprSetClr()</i> – set and clear bits in the DUART output port register <i>m68681Opr()</i> – return the current state of the DUART output port register <i>m68681OpcrSetClr()</i> – set and clear bits in the DUART output port configuration register <i>m68681Opcr()</i> – return the state of the DUART output port configuration register <i>m68681Int()</i> – handle all DUART interrupts in one vector |
| DESCRIPTION | <p>This is the driver for the M68681 DUART. This device includes two universal asynchronous receiver/transmitters, a baud rate generator, and a counter/timer device. This driver module provides control of the two serial channels and the baud-rate generator. The counter timer is controlled by a separate driver, src/drv/timer/m68681Timer.c.</p> <p>A M68681_DUART structure is used to describe the chip. This data structure contains two M68681_CHAN structures which describe the chip's two serial channels. The M68681_DUART structure is defined in m68681Sio.h.</p> <p>Only asynchronous serial operation is supported by this driver. The default serial settings are 8 data bits, 1 stop bit, no parity, 9600 baud, and software flow control. These default settings can be overridden on a channel-by-channel basis by setting the M68681_CHAN options and baudRate fields to the desired values before calling <i>m68681DevInit()</i>. See sioLib.h for option values. The defaults for the module can be changed by redefining the macros M68681_DEFAULT_OPTIONS and M68681_DEFAULT_BAUD and recompiling this driver.</p> <p>This driver supports baud rates of 75, 110, 134.5, 150, 300, 600, 1200, 2000, 2400, 4800, 1800, 9600, 19200, and 38400.</p> |
| USAGE | <p>The BSP's <i>sysHwInit()</i> routine typically calls <i>sysSerialHwInit()</i> which initializes all the hardware addresses in the M68681_DUART structure before calling <i>m68681DevInit()</i>. This enables the chip to operate in polled mode, but not in interrupt mode. Calling <i>m68681DevInit2()</i> from the <i>sysSerialHwInit2()</i> routine allows interrupts to be enabled and interrupt-mode operation to be used.</p> <p>The following example shows the first part of the initialization thorough calling <i>m68681DevInit()</i>:</p> |

```
#include "drv/sio/m68681Sio.h"
M68681_DUART myDuart; /* my device structure */
#define MY_VEC (71) /* use single vector, #71 */
sysSerialHwInit()
{
    /* initialize the register pointers for portA */
    myDuart.portA.mr = M68681_MRA;
    myDuart.portA.sr = M68681_SRA;
    myDuart.portA.csr = M68681_CSRA;
    myDuart.portA.cr = M68681_CRA;
    myDuart.portA.rb = M68681_RHRA;
    myDuart.portA.tb = M68681_THRA;
    /* initialize the register pointers for portB */
    myDuart.portB.mr = M68681_MRB;
    ...
    /* initialize the register pointers/data for main duart */
    myDuart.ivr = MY_VEC;
    myDuart.ipcr = M68681_IPCR;
    myDuart.acr = M68681_ACR;
    myDuart.isr = M68681_ISR;
    myDuart.imr = M68681_IMR;
    myDuart.ip = M68681_IP;
    myDuart.opcr = M68681_OPCR;
    myDuart.sopbc = M68681_SOPBC;
    myDuart.ropbc = M68681_ROPBC;
    myDuart.ctroff = M68681_CTROFF;
    myDuart.ctrton = M68681_CTRON;
    myDuart.ctrlr = M68681_CTRLR;
    myDuart.ctur = M68681_CTUR;
    m68681DevInit (&myDuart);
}
```

The BSP's *sysHwInit2()* routine typically calls *sysSerialHwInit2()* which connects the chips interrupts via *intConnect()* to the single interrupt handler *m68681Int()*. After the interrupt service routines are connected, the user then calls *m68681DevInit2()* to allow the driver to turn on interrupt enable bits, as shown in the following example:

```
sysSerialHwInit2 ()
{
    /* connect single vector for 68681 */
    intConnect (INUM_TO_IVEC(MY_VEC), m68681Int, (int)&myDuart);
    ...
    /* allow interrupts to be enabled */
    m68681DevInit2 (&myDuart);
}
```

SPECIAL CONSIDERATIONS

The CLOCAL hardware option presumes that OP0 and OP1 output bits are wired to the CTS outputs for channel 0 and channel 1 respectively. If not wired correctly, then the user must not select the CLOCAL option. CLOCAL is not one of the default options for this reason.

This driver does not manipulate the output port or its configuration register in any way. If the user selects the CLOCAL option, then the output port bit must be wired correctly or the hardware flow control will not function correctly.

INCLUDE FILES `drv/sio/m68681Sio.h`

m68901Sio

NAME `m68901Sio` – MC68901 MFP tty driver

ROUTINES `m68901DevInit()` – initialize a `M68901_CHAN` structure

DESCRIPTION This is the SIO driver for the Motorola MC68901 Multi-Function Peripheral (MFP) chip.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. However, one routine must be called directly: `m68901DevInit()` initializes the driver. Normally, it is called by `sysSerialHwInit()` in `sysSerial.c`

IOCTL FUNCTIONS This driver responds to the same `ioctl()` codes as other tty drivers; for more information, see the manual entry for `tyLib`.

SEE ALSO `tyLib`

masterIoLib

NAME `masterIoLib` – default IO routines for the SNMP master agent

ROUTINES `masterIoInit()` – create the IPC mechanism at the SNMP master agent
`snmpMonitorSpawn()` – spawn `tMonQue` to run `snmpQueMonitor()`
`masterIpcComp()` – transmit a completion of transmission message
`masterIoWrite()` – send the encoded buffer to the subagent
`masterIpcSend()` – send a message to a subagent
`masterIpcRcv()` – wait for a reply from the subagent

masterIpcAyt() – check the status of the IPC link
masterIpcFree() – free the IPC resources allocated by the SNMP master agent
masterQueCleanup() – free resources allocated for SNMP master agent

DESCRIPTION

This module implements the I/O routines used by the SNMP master agent. As shipped, the WindNet SNMP code uses message queues to communicate between the master agent and its subagents. The SNMP master agent also uses a message queue to handle communication between its two component tasks, **tSnmpd** and **tMonQue**. The **tSnmpd** task handles communication with the SNMP manager. The **tMonQue** task is a secondary task spawned from **tSnmpd** to receive messages from subagents.

When **tSnmpd** spawns **tMonQue**, it assigns *snmpQueMonitor()* to manage the process. This function waits on the message queue that subagents use to send messages to the master agent. The *snmpQueMonitor()* function interprets messages on its queue using an **SA_MESSAGE_T** structure, which is defined in **ipcLib.h** as:

```
typedef struct SA_MESSAGE_S
{
    int          msgType;
    MSG_Q_ID    saId;
    EBUFFER_T   msg;
} SA_MESSAGE_T;
```

A switch internal to *snmpQueMonitor()* handles the message according to the value of the **msgType** member.

If the message type is **CALL_QUERY_HANDLER**, the message is a response to a query from the master agent. The buffer referenced in the **msg** is then transferred to the local message queue monitored by **tSnmpd**, which is waiting for a query response from a subagent.

If the message type is **CALL_REG_HANDLER**, the message is either a registration request, a deregistration request, or some other control message (such as a trap). To respond to such requests, *snmpQueMonitor()* passes the buffer in **msg** to *snmpMasterHandlerWR()*.

If the message submitted to *snmpMasterHandlerWR()* is a registration request, it includes information on a set of leaves representing the objects that the subagent wants to add to the master agent's MIB tree. If the message passes all checks, the objects are added to the master agent's MIB tree and *snmpMasterHandlerWR()* returns success. All objects registered in one message become part of a group. They share the same IPC information, and, if the IPC link to their subagent is broken, they are deactivated as a group.

If *snmpMasterHandlerWR()* returns a function value indicating success, it also returns a message for the subagent containing the group ID for the variables just added. The *snmpQueMonitor()* takes responsibility for forwarding this message to the subagent. The subagent uses the group ID contained in this message when it comes time to deregister, as well as when it must register instances of an already registered object.

The returned function value of *snmpMasterHandlerWR()* could indicate failure or an opcode. You might want to rewrite this code to do something different. For example, if the

subagent had sent a trap up to the master agent, the returned value of *snmpMasterHandlerWR()* would be `SA_TRAP_REQUEST`, and the *vblst* parameter would contain a varbind list from the subagent. In this case, you would want to modify *snmpQueMonitor()* to pass the trap on to the SNMP manager.

MIB VARIABLES ADDED BY A SUBAGENT

These MIB variables that the subagent adds to the master agent's MIB tree look transparent to the SNMP manager that is in communication with the master agent. However, the method routines associated with these MIB variables in the master agent are not standard MIB routines. Instead, they are special routines that know how to queue test, get, and set work on the subagent that registered the MIB variables with the master agent. From the point of view of the PDU processing code, these special method routines look like any other method routines. However, when *tSnmpd* executes one of these routines, the special method routine actually passes the work on to a subagent while *tSnmpd* waits on a local message queue.

Because the subagent does not know about this local message queue, its response to the master agent is somewhat indirect. The only master agent message queue known to the subagent is the message queue managed by *tMonQue*, so the subagent puts its response on that queue. When the *snmpQueMonitor()* function that *tMonQue* runs to monitor the message queue sees that the message is a query response, it then transfers the message to the local queue upon which *tSnmpd* is awaiting a response. When *tSnmpd* sees the response, it parses it and merges the message into the PDU processing system.

SERIAL VERSUS ASYNCHRONOUS SUBAGENT PROCESSING

In the shipped implementation, communication between the master agent and its subagents is handled serially. For example, if the SNMP manager made a request concerning three MIB variables managed by three different subagents, the master agent would query each subagent in turn. After gathering all three responses, the master agent would then pack them up and ship the information back to the SNMP manager.

With some modifications to the code, you could rewrite the SNMP master agent to query all three subagents simultaneously (see the description of the *snmpMasterHandlerAsync()* function defined in *subagentLib.c*). That is, the master agent would query all three subagents one after the other without waiting for a response after making each request. If the subagents reside on different targets (each with its own processor), this asynchronous query method of multiple subagents lets you take advantage of the capacity for parallel processing.

However, if the subagents reside on different targets, you will also need to replace the code that implements the IPC mechanism used between the master agent and its subagents. In the shipped code, message queues serve as the IPC mechanism. To support agents that reside on different machines, you must replace this IPC mechanism with something such as sockets. To make this possible, the functions that implement the IPC mechanism are isolated to *masterIoLib.c* and *saIoLib.c*. These files ship as source code that you should feel free to edit as needed.

mathALib

| | |
|-----------------|---|
| NAME | mathALib – C interface library to high-level math functions |
| ROUTINES | <p><i>acos()</i> – compute an arc cosine (ANSI) <i>asin()</i> – compute an arc sine (ANSI) <i>atan()</i> – compute an arc tangent (ANSI) <i>atan2()</i> – compute the arc tangent of y/x (ANSI) <i>cbirt()</i> – compute a cube root <i>ceil()</i> – compute the smallest integer greater than or equal to a specified value (ANSI) <i>cos()</i> – compute a cosine (ANSI) <i>cosh()</i> – compute a hyperbolic cosine (ANSI) <i>exp()</i> – compute an exponential value (ANSI) <i>fabs()</i> – compute an absolute value (ANSI) <i>floor()</i> – compute the largest integer less than or equal to a specified value (ANSI) <i>fmod()</i> – compute the remainder of x/y (ANSI) <i>infinity()</i> – return a very large double <i>rint()</i> – convert a double-precision value to an integer <i>iround()</i> – round a number to the nearest integer <i>log()</i> – compute a natural logarithm (ANSI) <i>log10()</i> – compute a base-10 logarithm (ANSI) <i>log2()</i> – compute a base-2 logarithm <i>pow()</i> – compute the value of a number raised to a specified power (ANSI) <i>round()</i> – round a number to the nearest integer <i>sin()</i> – compute a sine (ANSI) <i>sincos()</i> – compute both a sine and cosine <i>sinh()</i> – compute a hyperbolic sine (ANSI) <i>sqrt()</i> – compute a non-negative square root (ANSI) <i>tan()</i> – compute a tangent (ANSI) <i>tanh()</i> – compute a hyperbolic tangent (ANSI) <i>trunc()</i> – truncate to integer <i>acosf()</i> – compute an arc cosine (ANSI) <i>asinf()</i> – compute an arc sine (ANSI) <i>atanf()</i> – compute an arc tangent (ANSI) <i>atan2f()</i> – compute the arc tangent of y/x (ANSI) <i>cbirtf()</i> – compute a cube root <i>ceilf()</i> – compute the smallest integer greater than or equal to a specified value (ANSI) <i>cosf()</i> – compute a cosine (ANSI) <i>coshf()</i> – compute a hyperbolic cosine (ANSI) <i>expf()</i> – compute an exponential value (ANSI) <i>fabsf()</i> – compute an absolute value (ANSI) <i>floorf()</i> – compute the largest integer less than or equal to a specified value (ANSI) <i>fmodf()</i> – compute the remainder of x/y (ANSI) <i>infinityf()</i> – return a very large float</p> |

rintf() – convert a single-precision value to an integer
roundf() – round a number to the nearest integer
logf() – compute a natural logarithm (ANSI)
log10f() – compute a base-10 logarithm (ANSI)
log2f() – compute a base-2 logarithm
powf() – compute the value of a number raised to a specified power (ANSI)
roundf() – round a number to the nearest integer
sinf() – compute a sine (ANSI)
sincosf() – compute both a sine and cosine
sinhf() – compute a hyperbolic sine (ANSI)
sqrtf() – compute a non-negative square root (ANSI)
tanf() – compute a tangent (ANSI)
tanhf() – compute a hyperbolic tangent (ANSI)
truncf() – truncate to integer

DESCRIPTION This library provides a C interface to high-level floating-point math functions, which can use either a hardware floating-point unit or a software floating-point emulation library. The appropriate routine is called based on whether *mathHardInit()* or *mathSoftInit()* or both have been called to initialize the interface.

All angle-related parameters are expressed in radians. All functions in this library with names corresponding to ANSI C specifications are ANSI compatible.

WARNING Not all functions in this library are available on all architectures. The architecture-specific appendices of the *VxWorks Programmer's Guide* list any math functions that are not available.

INCLUDE FILES **math.h**

SEE ALSO **ansiMath**, **fppLib**, **floatLib**, **mathHardLib**, **mathSoftLib**, Kernighan & Ritchie: *The C Programming Language*, 2nd Edition, *VxWorks Programmer's Guide: Architecture-specific Appendices*

mathHardLib

| | |
|----------------------|--|
| NAME | mathHardLib – hardware floating-point math library |
| ROUTINES | <i>mathHardInit()</i> – initialize hardware floating-point math support |
| DESCRIPTION | <p>This library provides support routines for using hardware floating-point units with high-level math functions. The high-level functions include trigonometric operations, exponents, and so forth.</p> <p>The routines in this library are used automatically for high-level math functions only if <i>mathHardInit()</i> has been called previously.</p> |
| WARNING | Not all architectures support hardware floating-point. See the architecture-specific appendices of the <i>VxWorks Programmer's Guide</i> . |
| INCLUDE FILES | math.h |
| SEE ALSO | mathSoftLib , mathALib , <i>VxWorks Programmer's Guide</i> architecture-specific appendices |

mathSoftLib

| | |
|----------------------|---|
| NAME | mathSoftLib – high-level floating-point emulation library |
| ROUTINES | <i>mathSoftInit()</i> – initialize software floating-point math support |
| DESCRIPTION | <p>This library provides software emulation of various high-level floating-point operations. This emulation is generally for use in systems that lack a floating-point coprocessor.</p> |
| WARNING | Software floating point is not supported for all architectures. See the architecture-specific appendices of the <i>VxWorks Programmer's Guide</i> . |
| INCLUDE FILES | math.h |
| SEE ALSO | mathHardLib , mathALib , <i>VxWorks Programmer's Guide</i> architecture-specific appendices |

mb86940Sio

| | |
|------------------------|--|
| NAME | mb86940Sio – MB 86940 UART tty driver |
| ROUTINES | <i>mb86940DevInit()</i> – install the driver function table |
| DESCRIPTION | This is the driver for the SPARClite MB86930 on-board serial ports. |
| USAGE | A MB86940_CHAN structure is used to describe the chip. The BSP's <i>sysHwInit()</i> routine typically calls <i>sysSerialHwInit()</i> , which initializes all the values in the MB86940_CHAN structure (except the SIO_DRV_FUNCS) before calling <i>mb86940DevInit()</i> . The BSP's <i>sysHwInit2()</i> routine typically calls <i>sysSerialHwInit2()</i> , which connects the chips interrupts via <i>intConnect()</i> . |
| IOCTL FUNCTIONS | The UARTs use timer 3 output to generate the following baud rates: 110, 150, 300, 600, 1200, 2400, 4800, 9600, and 19200. Note that the UARTs will operate at the same baud rate. |
| INCLUDE FILES | drv/sio/mb86940Sio.h |

mb86960End

| | |
|---------------------|---|
| NAME | mb86960End – END-style Fujitsu MB86960 Ethernet network interface driver |
| ROUTINES | <i>mb86960EndLoad()</i> – initialize the driver and device <i>mb86960InitParse()</i> – parse the initialization string <i>mb86960MemInit()</i> – initialize memory for the chip |
| DESCRIPTION | This module implements the Fujitsu MB86960 NICE Ethernet network interface driver. This driver is non-generic and has only been run on the Fujitsu SPARClite Evaluation Board. It currently supports only unit number zero. The driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below. |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. The MB86960 Network Interface Controller with Encoder/Decoder (NICE) chip is a highly integrated monolithic device which incorporates both network controller, complete with buffer management and Manchester encoder/decoder. |

TARGET-SPECIFIC PARAMETERS

The format of the parameter string is *unit:devBaseAddr:ivec*, where:

unit

A convenient holdover from the former model. It is only used in the string name for the driver.

devBaseAddr

The base Address of the chip registers.

ivec

This is the interrupt vector number of the hardware interrupt generated by this ethernet device. The driver uses *intConnect()* to attach an interrupt handler to this interrupt.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires seven external support functions:

sys86960IntEnable()

void sysEnetIntEnable (int unit)

This routine provides a target-specific interface to enable Ethernet device interrupts for a given device unit. For this driver, value of unit must be 0.

sys86960IntDisable()

void sysEnetIntDisable (int unit)

This routine provides a target-specific interface to disable Ethernet device interrupts for a given device unit. For this driver, value of unit must be 0.

sysEnetAddrGet()

STATUS sysEnetAddrGet (int unit, char *enetAdrs)

This routine provides a target-specific interface to access a device Ethernet address. This routine should provide a six-byte Ethernet address in the *enetAdrs* parameter and return OK or ERROR.

In this driver the macros **SYS_OUT_SHORT** and **SYS_IN_SHORT** which call BSP-specific functions to access the chip register.

INCLUDES **end.h endLib.h etherMultiLib.h**

SEE ALSO **muxLib, endLib, Writing and Enhanced Network Driver**

mb87030Lib

| | |
|-------------------------------|---|
| NAME | mb87030Lib – Fujitsu MB87030 SCSI Protocol Controller (SPC) library |
| ROUTINES | <i>mb87030CtrlCreate()</i> – create a control structure for an MB87030 SPC <i>mb87030CtrlInit()</i> – initialize a control structure for an MB87030 SPC <i>mb87030Show()</i> – display the values of all readable MB87030 SPC registers |
| DESCRIPTION | This is the I/O driver for the Fujitsu MB87030 SCSI Protocol Controller (SPC) chip. It is designed to work in conjunction with scsiLib . |
| USER-CALLABLE ROUTINES | Most of the routines in this driver are accessible only through the I/O system. Two routines, however, must be called directly: <i>mb87030CtrlCreate()</i> to create a controller structure, and <i>mb87030CtrlInit()</i> to initialize the controller structure. |
| INCLUDE FILES | mb87030.h |
| SEE ALSO | scsiLib , <i>Fujitsu Small Computer Systems Interface MB87030 Synchronous/Asynchronous Protocol Controller Users Manual</i> , <i>VxWorks Programmer's Guide: I/O System</i> |

mbcEnd

| | |
|--------------------|---|
| NAME | mbcEnd – Motorola 68302fads END network interface driver |
| ROUTINES | <i>mbcEndLoad()</i> – initialize the driver and device <i>mbcParse()</i> – parse the init string <i>mbcMemInit()</i> – initialize memory for the chip <i>mbcAddrFilterSet()</i> – set the address filter for multicast addresses |
| DESCRIPTION | This is a driver for the Ethernet controller on the 68EN302 chip. The device supports a 16-bit interface, data rates up to 10 Mbps, a dual-ported RAM, and transparent DMA. The dual-ported RAM is used for a 64-entry CAM table, and a 128-entry buffer descriptor table. The CAM table is used to set the Ethernet address of the Ethernet device or to program multicast addresses. The buffer descriptor table is partitioned into fixed-size transmit and receive tables. The DMA operation is transparent and transfers data between the internal FIFOs and external buffers pointed to by the receive and transmit-buffer descriptors during transmits and receives. The driver requires that the memory used for transmit and receive buffers be allocated in cache-safe RAM area. |

Up to 61 multicast addresses are supported. Multicast addresses are supported by adding the multicast ethernet addresses to the address table in the ethernet part. If more than 61 multicast addresses are desired, address hashing must be used (the address table holds 62 entries at most). However, address hashing does not appear to work in this ethernet part.

A glitch in the EN302 Rev 0.1 device causes the Ethernet transmitter to lock up from time to time. The driver uses a watchdog timer to reset the Ethernet device when the device runs out of transmit buffers and cannot recover within 20 clock ticks.

BOARD LAYOUT This device is on-chip. No jumpering diagram is necessary.

EXTERNAL INTERFACE

The only external interface is the *mbcEndLoad()* routine, which expects the *initString* parameter as input. This parameter passes in a colon-delimited string of the format:

unit:memAddr:ivec:txBdNum:rxBdNum:dmaParms:bufBase:offset

TARGET-SPECIFIC PARAMETERS

unit

A convenient holdover from the former model. This parameter is used only in the string name for the driver.

memAddr

This parameter is the base address of the Ethernet module. The driver addresses all other Ethernet device registers as offsets from this address.

ivec

The interrupt vector to be used in connecting the interrupt handler.

txBdNum

The number of transmit buffer descriptors to use.

rxBdNum

The number of receive buffer descriptors to use.

The number of transmit and receive buffer descriptors (BDs) used is configurable by the user while attaching the driver. Each BD is 8 bytes in size and resides in the chip's dual-ported memory, while its associated buffer, 1520 bytes in size, resides in cache-safe conventional RAM. A minimum of 2 receive and 2 transmit BDs should be allocated. If this parameter is 0, a default of 32 BDs will be used. The maximum number of BDs depends on how the dual-ported BD RAM is partitioned. The 128 BDs in the dual-ported BD RAM can be partitioned into transmit and receive BD regions with 8, 16, 32, or 64 transmit BDs and corresponding 120, 112, 96, or 64 receive BDs.

dmaParms

Ethernet DMA parameters.

This parameter is used to specify the settings of burst limit, water-mark, and transmit early, which control the Ethernet DMA, and is used to set the EDMA register.

bufBase

Base address of the buffer pool.

This parameter is used to notify the driver that space for the transmit and receive buffers need not be allocated, but should be taken from a cache-coherent private memory space provided by the user at the given address. The user should be aware that memory used for buffers must be 4-byte aligned and non-cacheable. All the buffers must fit in the given memory space; no checking will be performed. Each buffer is 1520 bytes. If this parameter is "NULL", space for buffers will be obtained by calling *cacheDmaMalloc()* in *mbcMemInit()*.

offset

Specifies the memory alignment offset.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires several external support functions, defined as macros:

```
SYS_INT_CONNECT(pDrvCtrl, routine, arg)
SYS_INT_DISCONNECT (pDrvCtrl, routine, arg)
SYS_INT_ENABLE(pDrvCtrl)
SYS_OUT_SHORT(pDrvCtrl, reg, data)
SYS_IN_SHORT(pDrvCtrl, reg, pData)
```

There are default values in the source code for these macros. They presume memory-mapped accesses to the device registers and the normal *intConnect()*, and *intEnable()* BSP functions. The first argument to each is the device controller structure. Thus, each has access back to all the device-specific information. Having the pointer in the macro facilitates the addition of new features to this driver.

SYSTEM RESOURCE USAGE

The driver requires the following system resources:

- one watchdog timer
- one interrupt vector
- 52 bytes in the initialized data section (data)
- 0 bytes in the uninitialized data section (bss)

The above data and bss requirements are for MC680x0 architectures and varies for other architectures. Code size (text) varies greatly between architectures and is not quoted here.

If the driver allocates the memory shared with the Ethernet device unit, it does so by calling the *cacheDmaMalloc()* routine. For the default case of 32 transmit buffers, 32 receive buffers, the total size requested is roughly 100,000 bytes. If a memory region is provided to the driver, the size of this region is adjustable to suit user needs.

This driver can only operate if the shared memory region is non-cacheable, or if the hardware implements bus snooping. The driver cannot maintain cache coherency for the device because the buffers are asynchronously modified by both the driver and the device, and these fields may share the same cache line. Additionally, the chip's dual-ported RAM must be declared as non-cacheable memory where applicable.

| | |
|-----------------|--|
| INCLUDES | end.h endLib.h etherMultiLib.h |
| SEE ALSO | muxLib, endLib, Writing and Enhanced Network Driver |

memDrv

| | |
|-------------------------------|---|
| NAME | memDrv – pseudo memory device driver |
| ROUTINES | <p><i>memDrv()</i> – install a memory driver</p> <p><i>memDevCreate()</i> – create a memory device</p> <p><i>memDevCreateDir()</i> – create a memory device for multiple files</p> <p><i>memDevDelete()</i> – delete a memory device</p> |
| DESCRIPTION | <p>This driver allows the I/O system to access memory directly as a pseudo-I/O device. Memory location and size are specified when the device is created. This feature is useful when data must be preserved between boots of VxWorks or when sharing data between CPUs.</p> <p>Additionally, it can be used to build some files into a VxWorks binary image (having first converted them to data arrays in C source files, using a utility such as <i>memdrvbuild</i>), and then mount them in the filesystem; this is a simple way of delivering some non-changing files with VxWorks. For example, a system with an integrated web server may use this technique to build some HTML and associated content files into VxWorks.</p> <p>memDrv can be used to simply provide a high-level method of reading and writing bytes in absolute memory locations through I/O calls. It can also be used to implement a simple, essentially read-only filesystem (existing files can be rewritten within their existing sizes); directory searches and a limited set of IOCTL calls (including <i>stat()</i>) are supported.</p> |
| USER-CALLABLE ROUTINES | <p>Most of the routines in this driver are accessible only through the I/O system. Four routines, however, can be called directly: <i>memDrv()</i> to initialize the driver, <i>memDevCreate()</i> and <i>memDevCreateDir()</i> to create devices, and <i>memDevDelete()</i> to delete devices.</p> <p>Before using the driver, it must be initialized by calling <i>memDrv()</i>. This routine should be called only once, before any reads, writes, or <i>memDevCreate()</i> calls. It may be called from <i>usrRoot()</i> in <i>usrConfig.cor</i> at some later point.</p> |
| IOCTL FUNCTIONS | The dosFs file system supports the following <i>ioctl()</i> functions. The functions listed are defined in the header ioLib.h . Unless stated otherwise, the file descriptor used for these functions may be any file descriptor which is opened to a file or directory on the volume or to the volume itself. |

FIOGETFL

Copies to *flags* the open mode flags of the file (`O_RDONLY`, `O_WRONLY`, `O_RDWR`):

```
int flags;
status = ioctl (fd, FIOGETFL, &flags);
```

FIOSEEK

Sets the current byte offset in the file to the position specified by *newOffset*:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

The `FIOSEEK` offset is always relative to the beginning of the file. The offset, if any, given at open time by using pseudo-file name is overridden.

FIOWHERE

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

FIONREAD

Copies to *unreadCount* the number of unread bytes in the file:

```
int unreadCount;
status = ioctl (fd, FIONREAD, &unreadCount);
```

FIOREADDIR

Reads the next directory entry. The argument *dirStruct* is a `DIR` directory descriptor. Normally, the `readdir()` routine is used to read a directory, rather than using the `FIOREADDIR` function directly. See `dirLib`.

```
DIR dirStruct;
fd = open ("directory", O_RDONLY);
status = ioctl (fd, FIOREADDIR, &dirStruct);
```

FIOFSTATGET

Gets file status information (directory entry data). The argument *statStruct* is a pointer to a `stat` structure that is filled with data describing the specified file. File inode numbers, user and group IDs, and times are not supported (returned as 0).

Normally, the `stat()` or `fstat()` routine is used to obtain file information, rather than using the `FIOFSTATGET` function directly. See `dirLib`.

```
struct stat statStruct;
fd = open ("file", O_RDONLY);
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

Any other `ioctl()` function codes will return error status.

SEE ALSO

VxWorks Programmer's Guide: I/O System

memLib

| | |
|----------------------|--|
| NAME | memLib – full-featured memory partition manager |
| ROUTINES | <i>memPartOptionsSet()</i> – set the debug options for a memory partition <i>memalign()</i> – allocate aligned memory <i>valloc()</i> – allocate memory on a page boundary <i>memPartRealloc()</i> – reallocate a block of memory in a specified partition <i>memPartFindMax()</i> – find the size of the largest available free block <i>memOptionsSet()</i> – set the debug options for the system memory partition <i>calloc()</i> – allocate space for an array (ANSI) <i>realloc()</i> – reallocate a block of memory (ANSI) <i>cfree()</i> – free a block of memory <i>memFindMax()</i> – find the largest free block in the system memory partition |
| DESCRIPTION | <p>This library provides full-featured facilities for managing the allocation of blocks of memory from ranges of memory called memory partitions. The library is an extension of memPartLib and provides enhanced memory management features, including error handling, aligned allocation, and ANSI allocation routines. For more information about the core memory partition management facility, see the manual entry for memPartLib.</p> <p>The system memory partition is created when the kernel is initialized by <i>kernelInit()</i>, which is called by the root task, <i>usrRoot()</i>, in usrConfig.c. The ID of the system memory partition is stored in the global variable memSysPartId; its declaration is included in memLib.h.</p> <p>The <i>memalign()</i> routine is provided for allocating memory aligned to a specified boundary.</p> <p>This library includes three ANSI-compatible routines: <i>calloc()</i> allocates a block of memory for an array; <i>realloc()</i> changes the size of a specified block of memory; and <i>cfree()</i> returns to the free memory pool a block of memory that was previously allocated with <i>calloc()</i>.</p> |
| ERROR OPTIONS | <p>Various debug options can be selected for each partition using <i>memPartOptionsSet()</i> and <i>memOptionsSet()</i>. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, the error status is returned. There are four error-handling options that can be individually selected:</p> <p>MEM_ALLOC_ERROR_LOG_FLAG Log a message when there is an error in allocating memory.</p> <p>MEM_ALLOC_ERROR_SUSPEND_FLAG Suspend the task when there is an error in allocating memory (unless the task was spawned with the VX_UNBREAKABLE option, in which case it cannot be suspended).</p> |

MEM_BLOCK_ERROR_LOG_FLAG

Log a message when there is an error in freeing memory.

MEM_BLOCK_ERROR_SUSPEND_FLAG

Suspend the task when there is an error in freeing memory (unless the task was spawned with the **VX_UNBREAKABLE** option, in which case it cannot be suspended).

When the following option is specified to check every block freed to the partition, *memPartFree()* and *free()* in **memPartLib** run consistency checks of various pointers and values in the header of the block being freed. If this flag is not specified, no check will be performed when memory is freed.

MEM_BLOCK_CHECK

Check each block freed.

Setting either of the **MEM_BLOCK_ERROR** options automatically sets **MEM_BLOCK_CHECK**.

The default options when a partition is created are:

```
MEM_ALLOC_ERROR_LOG_FLAG
MEM_BLOCK_CHECK
MEM_BLOCK_ERROR_LOG_FLAG
MEM_BLOCK_ERROR_SUSPEND_FLAG
```

When setting options for a partition with *memPartOptionsSet()* or *memOptionsSet()*, use the logical OR operator between each specified option to construct the *options* parameter. For example:

```
memPartOptionsSet (myPartId, MEM_ALLOC_ERROR_LOG_FLAG |
                  MEM_BLOCK_CHECK |
                  MEM_BLOCK_ERROR_LOG_FLAG);
```

INCLUDE FILES **memLib.h**

SEE ALSO **memPartLib, smMemLib**

memPartLib

NAME **memPartLib** – core memory partition manager

ROUTINES *memPartCreate()* – create a memory partition
memPartAddToPool() – add memory to a memory partition
memPartAlignedAlloc() – allocate aligned memory from a partition
memPartAlloc() – allocate a block of memory from a partition
memPartFree() – free a block of memory in a partition
memAddToPool() – add memory to the system memory partition
malloc() – allocate a block of memory from the system memory partition (ANSI)
free() – free a block of memory (ANSI)

DESCRIPTION This library provides core facilities for managing the allocation of blocks of memory from ranges of memory called memory partitions. The library was designed to provide a compact implementation; full-featured functionality is available with **memLib**, which provides enhanced memory management features built as an extension of **memPartLib**. (For more information about enhanced memory partition management options, see the manual entry for **memLib**.) This library consists of two sets of routines. The first set, *memPart...()*, comprises a general facility for the creation and management of memory partitions, and for the allocation and deallocation of blocks from those partitions. The second set provides a traditional ANSI-compatible *malloc()/free()* interface to the system memory partition.

The system memory partition is created when the kernel is initialized by *kernelInit()*, which is called by the root task, *usrRoot()*, in **usrConfig.c**. The ID of the system memory partition is stored in the global variable **memSysPartId**; its declaration is included in **memLib.h**.

The allocation of memory, using *malloc()* in the typical case and *memPartAlloc()* for a specific memory partition, is done with a first-fit algorithm. Adjacent blocks of memory are coalesced when they are freed with *memPartFree()* and *free()*. There is also a routine provided for allocating memory aligned to a specified boundary from a specific memory partition, *memPartAlignedAlloc()*.

CAVEATS Architectures have various alignment constraints. To provide optimal performance, *malloc()* returns a pointer to a buffer having the appropriate alignment for the architecture in use. The portion of the allocated buffer reserved for system bookkeeping, known as the overhead, may vary depending on the architecture.

| Architecture | Boundary | Overhead |
|--------------|----------|----------|
| 68K | 4 | 8 |
| SPARC | 8 | 12 |
| MIPS | 16 | 12 |
| i960 | 16 | 16 |

INCLUDE FILES **memLib.h, stdlib.h**

SEE ALSO **memLib, smMemLib**

memShow

NAME **memShow** – memory show routines

ROUTINES *memShowInit()* – initialize the memory partition show facility
memShow() – show system memory partition blocks and statistics
memPartShow() – show partition blocks and statistics
memPartInfoGet() – get partition information

DESCRIPTION This library contains memory partition information display routines. To use this facility, it must first be installed using *memShowInit()*, which is called automatically when the memory partition show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define **INCLUDE_SHOW_ROUTINES** in **config.h**.
- If you use the Tornado project facility, select **INCLUDE_MEM_SHOW**.

SEE ALSO **memLib, memPartLib**, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

mmanPxLib

NAME **mmanPxLib** – memory management library (POSIX)

ROUTINES *mlockall()* – lock all pages used by a process into memory (POSIX)
munlockall() – unlock all pages used by a process (POSIX)
mlock() – lock specified pages into memory (POSIX)
munlock() – unlock specified pages (POSIX)

DESCRIPTION This library contains POSIX interfaces designed to lock and unlock memory pages, i.e., to control whether those pages may be swapped to secondary storage. Since VxWorks does not use swapping (all pages are always kept in memory), these routines have no real effect and simply return 0 (OK).

INCLUDE FILES sys/mman.h

SEE ALSO POSIX 1003.1b document

mmuL64862Lib

NAME **mmuL64862Lib** – LSI Logic L64862 MBus-to-SBus Interface: I/O DMA library (SPARC)

ROUTINES *mmuL64862DmaInit()* – initialize the L64862 I/O MMU DMA data structures (SPARC)

DESCRIPTION This library contains the architecture-specific routine *mmuL64862DmaInit()*, needed to set up the I/O mapping for S-Bus DMA devices using the LSI Logic L64862 architecture.

INCLUDE FILES arch/sparc/l64862.h

SEE ALSO **cacheLib**, **vmLib**

mmuPro32Lib

NAME **mmuPro32Lib** – mmu library for PentiumPro/II

ROUTINES *mmuPro32LibInit()* – initialize module

DESCRIPTION **mmuPro32Lib.c** provides the architecture dependent routines that directly control the memory management unit. It provides 10 routines that are called by the higher level architecture independent routines in **vmLib.c**:

- mmuLibInit()* – initialize module
- mmuTransTblCreate()* – create a new translation table
- mmuTransTblDelete()* – delete a translation table.
- mmuEnable()* – turn MMU on or off
- mmuStateSet()* – set state of virtual memory page
- mmuStateGet()* – get state of virtual memory page
- mmuPageMap()* – map physical memory page to virtual memory page
- mmuGlobalPageMap()* – map physical memory page to global virtual memory page
- mmuTranslate()* – translate a virtual address to a physical address
- mmuCurrentSet()* – change active translation table

Applications using the MMU will never call these routines directly; the visible interface is supported in **vmLib.c**.

mmuLib supports the creation and maintenance of multiple translation tables, one of which is the active translation table when the mmu is enabled. Note that VxWorks does not include a translation table as part of the task context; individual tasks do not reside in private virtual memory. However, we include the facilities to create multiple translation tables so that the user may create "private" virtual memory contexts and switch them in an application specific manner. New translation tables are created with a call to *mmuTransTblCreate()*, and installed as the active translation table with *mmuCurrentSet()*. Translation tables are modified and potentially augmented with calls to *mmuPageMap()* and *mmuStateSet()*. The state of portions of the translation table can be read with calls to *mmuStateGet()* and *mmuTranslate()*.

The traditional VxWorks architecture and design philosophy requires that all objects and operating systems resources be visible and accessible to all agents (tasks, ISRs, watchdog timers, etc) in the system. This has traditionally been insured by the fact that all objects and data structures reside in physical memory; thus, a data structure created by one agent may be accessed by any other agent using the same pointer (object identifiers in VxWorks are often pointers to data structures.) This creates a potential problem if you have multiple virtual memory contexts. For example, if a semaphore is created in one virtual memory context, you must guarantee that that semaphore will be visible in all virtual memory contexts if the semaphore is to be accessed at interrupt level, when a virtual memory context other than the one in which it was created may be active. Another example is that code loaded using the incremental loader from the shell must be accessible in all virtual memory contexts, since code is shared by all agents in the system.

This problem is resolved by maintaining a global "transparent" mapping of virtual to physical memory for all the contiguous segments of physical memory (on board memory, I/O space, sections of VME space, etc.) that is shared by all translation tables; all available physical memory appears at the same address in virtual memory in all virtual memory contexts. This technique provides an environment that allows resources that rely on a globally accessible physical address to run without modification in a system with multiple virtual memory contexts.

An additional requirement is that modifications made to the state of global virtual memory in one translation table appear in all translation tables. For example, memory containing the text segment is made read only (to avoid accidental corruption) by setting the appropriate writeable bits in the translation table entries corresponding to the virtual memory containing the text segment. This state information must be shared by all virtual memory contexts, so that no matter what translation table is active, the text segment is protected from corruption. The mechanism that implements this feature is architecture dependent, but usually entails building a section of a translation table that corresponds to the global memory, that is shared by all other translation tables. Thus, when changes to the state of the global memory are made in one translation table, the changes are reflected in all other translation tables.

mmuLib provides a separate call for constructing global virtual memory – *mmuGlobalPageMap()* – which creates translation table entries that are shared by all translation tables. Initialization code in **usrConfig** makes calls to *vmGlobalMap()* (which

The physical memory that holds these data structures is obtained from the system memory manager via memalign to insure that the memory is page aligned. We want to protect this memory from being corrupted, so we invalidate the descriptors that we set up in the global translation that correspond to the memory containing the translation table data structures. This creates a "chicken and the egg" paradox, in that the only way we can modify these data structures is through virtual memory that is now invalidated, and we can't validate it because the page descriptors for that memory are in invalidated memory (confused yet?) So, you will notice that anywhere that page table descriptors (pte's) are modified, we do so by locking out interrupts, momentarily disabling the mmu, accessing the memory with its physical address, enabling the mmu, and then re-enabling interrupts (see `mmuStateSet()`, for example.)

Support for two new page attribute bits are added for PentiumPro's enhanced MMU. They are Global bit (G) and Page-level write-through/back bit (PWT). Global bit indicates a global page when set. When a page is marked global and the page global enable (PGE) bit in register CR4 is set, the page-table or page-directory entry for the page is not invalidated in the TLB when register CR3 is loaded or a task switch occurs. This bit is provided to prevent frequently used pages (such as pages that contain kernel or other operating system or executive code) from being flushed from the TLB. Page-level write-through/back bit (PWT) controls the write-through or write-back caching policy of individual pages or page tables. When the PWT bit is set, write-through caching is enabled for the associated page or page table. When the bit is clear, write-back caching is enabled for the associated page and page table. Following macros are used to describe these attribute bits in the physical memory descriptor table `sysPhysMemDesc[]` in `sysLib.c`.

```
VM_STATE_WBACK      = use write-back cache policy for the page
VM_STATE_WBACK_NOT  = use write-through cache policy for the page
VM_STATE_GLOBAL     = set page global bit
VM_STATE_GLOBAL_NOT = not set page global bit
```

Support for two page size (4KB and 4MB) are added also. The linear address for 4KB pages is divided into three sections:

```
Page directory entry - bits 22 through 31.
Page table entry     - Bits 12 through 21.
Page offset          - Bits 0 through 11.
```

The linear address for 4MB pages is divided into two sections:

```
Page directory entry - Bits 22 through 31.
Page offset          - Bits 0 through 21.
```

These two page size is configurable by `VM_PAGE_SIZE` macro in `config.h`.

mmuSparcILib

| | |
|--------------------|--|
| NAME | mmuSparcILib – ROM MMU initialization (SPARC) |
| ROUTINES | <i>mmuSparcRomInit()</i> – initialize the MMU for the ROM (SPARC) |
| DESCRIPTION | <p>This library contains routines that are called by SPARC boot ROMs to initialize the translation tables while still in "boot state." When the board comes up, all instruction fetches from the boot ROMs bypass the MMU, thus allowing code in the ROMs to initialize the MMU tables with mappings for RAM, I/O devices, and other memory devices.</p> <p><i>mmuSparcRomInit()</i> is called from <i>romInit()</i>. The translation tables are initialized according to the mappings found in sysPhysMemDesc, which is contained in memDesc.c in the BSP. Note that these mappings are also used by vmLib or vmBaseLib when VxWorks creates global virtual memory at system initialization time. New ROMs may need to be built if these tables are modified.</p> |

moduleLib

| | |
|--------------------|---|
| NAME | moduleLib – object module management library |
| ROUTINES | <p><i>moduleCreate()</i> – create and initialize a module <i>moduleDelete()</i> – delete module ID information (use <i>unld()</i> to reclaim space) <i>moduleShow()</i> – show the current status for all the loaded modules <i>moduleSegGet()</i> – get (delete and return) the first segment from a module <i>moduleSegFirst()</i> – find the first segment in a module <i>moduleSegNext()</i> – find the next segment in a module <i>moduleCreateHookAdd()</i> – add a routine to be called when a module is added <i>moduleCreateHookDelete()</i> – delete a previously added module create hook routine <i>moduleFindByName()</i> – find a module by name <i>moduleFindByNameAndPath()</i> – find a module by file name and path <i>moduleFindByGroup()</i> – find a module by group number <i>moduleIdListGet()</i> – get a list of loaded modules <i>moduleInfoGet()</i> – get information about an object module <i>moduleCheck()</i> – verify checksums on all modules <i>moduleNameGet()</i> – get the name associated with a module ID <i>moduleFlagsGet()</i> – get the flags associated with a module ID</p> |
| DESCRIPTION | <p>This library is a class manager, using the standard VxWorks class/object facilities. The library is used to keep track of which object modules have been loaded into VxWorks, to</p> |

motCpmEnd

maintain information about object module segments associated with each module, and to track which symbols belong to which module. Tracking modules makes it possible to list which modules are currently loaded, and to unload them when they are no longer needed.

The module object contains the following information:

- name
- linked list of segments, including base addresses and sizes
- symbol group number
- format of the object module (a.out, COFF, ECOFF, etc.)
- the *symFlag* passed to *ld()* when the module was loaded. (For more information about *symFlag* and the loader, see the manual entry for *loadLib*.)

Multiple modules with the same name are allowed (the same module may be loaded without first being unloaded) but "find" functions find the most recently created module.

The symbol group number is a unique number for each module, used to identify the module's symbols in the symbol table. This number is assigned by **moduleLib** when a module is created.

In general, users will not access these routines directly, with the exception of *moduleShow()*, which displays information about currently loaded modules. Most calls to this library will be from routines in **loadLib** and **unldLib**.

INCLUDE FILES **moduleLib.h**

SEE ALSO **loadLib**

motCpmEnd

NAME **motCpmEnd** – END style Motorola MC68EN360/MPC800 network interface driver

ROUTINES *motCpmEndLoad()* – initialize the driver and device

DESCRIPTION This module implements the Motorola MC68EN360 QUICC as well as the MPC821 and MPC860 Power-QUICC Ethernet Enhanced network interface driver.

All the above mentioned microprocessors feature a number of Serial Communication Controllers (SCC) that support different serial protocols including IEEE 802.3 and Ethernet CSMA-CD. As a result, when the Ethernet mode of a SCC is selected, by properly programming its general Mode Register (GSMR), they can implement the full set of media access control and channel interface functions those protocol require. However, while the

MC68EN360 QUICC and the MPC860 Power-QUICC support up to four SCCs per unit, the MPC821 only includes two on-chip SCCs.

This driver is designed to support the Ethernet mode of a SCC residing on the CPM processor core, no matter which among the MC68EN360 QUICC or any of the PPC800 Series. In fact, the major differences among these processors, as far as the driver is concerned, are to be found in the mapping of the internal Dual-Port RAM. The driver is generic in the sense that it does not care which SCC is being used. In addition, it poses no constraint on the number of individual units that may be used per board. However, this number should be specified in the bsp through the macro `MAX_SCC_CHANNELS`. The default value for this macro in the driver is 4.

To achieve these goals, the driver requires several target-specific values provided as an input string to the load routine. It also requires some external support routines. These target-specific values and the external support routines are described below.

This network interface driver does not include support for trailer protocols or data chaining. However, buffer loaning has been implemented in an effort to boost performance.

This driver maintains cache coherency by allocating buffer space using the `cacheDmaMalloc()` routine. This is provided for boards whose host processor use data cache space, e.g. the MPC800 Series. Although the MC68EN360 does not have cache memory, it may be used in a particular configuration: **MC68EN360 in 040 companion mode** where that is attached to processors that may cache memory. However, due to a lack of suitable hardware, the multiple unit support and '040 companion mode support have not been tested.

BOARD LAYOUT This device is on-chip. No jumpering diagram is necessary.

EXTERNAL INTERFACE

This driver provides the standard END external interface. The only external interface is the `motCpmEndLoad()` routine. The parameters are passed into the `motCpmEndLoad()` function as a single colon-delimited string. The `motCpmEndLoad()` function uses `strtok()` to parse the string, which it expects to be of the following format:

```
unit:motCpmAddr:ivec:sccNum:txBdNum:rxBdNum: txBdBase: rxBdBase:bufBase
```

TARGET-SPECIFIC PARAMETERS

unit

A convenient holdover from the former model. This parameter is used only in the string name for the driver.

motCpmAddr

Indicates the address at which the host processor presents its internal memory (also known as the dual ported RAM base address). With this address, and the SCC number (see below), the driver is able to compute the location of the SCC parameter RAM and the SCC register map, and, ultimately, to program the SCC for proper

operations. This parameter should point to the internal memory of the processor where the SCC physically resides. This location might not necessarily be the Dual-Port RAM of the microprocessor configured as master on the target board.

ivec

This driver configures the host processor to generate hardware interrupts for various events within the device. The interrupt-vector offset parameter is used to connect the driver's ISR to the interrupt through a call to the VxWorks system function *intConnect()*.

sccNum

This driver is written to support multiple individual device units. Thus, the multiple units supported by this driver can reside on different chips or on different SCCs within a single host processor. This parameter is used to explicitly state which SCC is being used (SCC1 is most commonly used, thus this parameter most often equals "1").

txBdNum and *rxBdNum*

Specify the number of transmit and receive buffer descriptors (BDs). Each buffer descriptor resides in 8 bytes of the processor's dual-ported RAM space, and each one points to a 1520 byte buffer in regular RAM. There must be a minimum of two transmit and two receive BDs. There is no maximum, although more than a certain amount does not speed up the driver and wastes valuable dual-ported RAM space. If any of these parameters is "NULL", a default value of "32" BDs is used.

txBdBase and *rxBdBase*

Indicate the base location of the transmit and receive buffer descriptors (BDs). They are offsets, in bytes, from the base address of the host processor's internal memory (see above). Each BD takes up 8 bytes of dual-ported RAM, and it is the user's responsibility to ensure that all specified BDs fit within dual-ported RAM. This includes any other BDs the target board might be using, including other SCCs, SMCs, and the SPI device. There is no default for these parameters. They must be provided by the user.

bufBase

Tells the driver that space for the transmit and receive buffers need not be allocated but should be taken from a cache-coherent private memory space provided by the user at the given address. The user should be aware that memory used for buffers must be 4-byte aligned and non-cacheable. All the buffers must fit in the given memory space. No checking is performed. This includes all transmit and receive buffers (see above). Each buffer is 1520 bytes. If this parameter is "NONE", space for buffers is obtained by calling *cacheDmaMalloc()* in *motCpmEndLoad()*.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires three external support functions:

sysXxxEnetEnable()

This is either *sys360EnetEnable()* or *sysCpmEnetEnable()*, based on the actual host processor being used. See below for the actual prototypes. This routine is expected to

handle any target-specific functions needed to enable the Ethernet controller. These functions typically include enabling the Transmit Enable signal (TENA) and connecting the transmit and receive clocks to the SCC. This routine is expected to return OK on success, or ERROR. The driver calls this routine, once per unit, from the *motCpmEndLoad()* routine.

sysXxxEnetDisable()

This is either *sys360EnetDisable()* or *sysCpmEnetDisable()*, based on the actual host processor being used. See below for the actual prototypes. This routine is expected to handle any target-specific functions required to disable the Ethernet controller. This usually involves disabling the Transmit Enable (TENA) signal. This routine is expected to return OK on success, or ERROR. The driver calls this routine from the *motCpmEndStop()* routine each time a unit is disabled.

sysXxxEnetAddrGet()

This is either *sys360EnetAddrGet()* or *sysCpmEnetAddrGet()*, based on the actual host processor being used. See below for the actual prototypes. The driver expects this routine to provide the six-byte Ethernet hardware address that is used by this unit. This routine must copy the six-byte address to the space provided by *addr*. This routine is expected to return OK on success, or ERROR. The driver calls this routine, once per unit, from the *motCpmEndLoad()* routine.

In the case of the CPU32, the prototypes of the above mentioned support routines are as follows:

```
STATUS sys360EnetEnable (int unit, UINT32 regBase)
void sys360EnetDisable (int unit, UINT32 regBase)
STATUS sys360EnetAddrGet (int unit, u_char * addr)
```

In the case of the PPC860, the prototypes of the above mentioned support routines are as follows:

```
STATUS sysCpmEnetEnable (int unit)
void sysCpmEnetDisable (int unit)
STATUS sysCpmEnetAddrGet (int unit, UINT8 * addr)
```

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- 0 bytes in the initialized data section (data)
- 1272 bytes in the uninitialized data section (BSS)

The data and BSS sections are quoted for the CPU32 architecture and could vary for other architectures. The code size (text) varies greatly between architectures, and is therefore not quoted here.

If the driver allocates the memory to share with the Ethernet device unit, it does so by calling the *cacheDmaMalloc()* routine. For the default case of 32 transmit buffers, 32

motFecEnd

receive buffers, and 16 loaner buffers (this is not configurable), the total size requested is 121,600 bytes. If a non-cacheable memory region is provided by the user, the size of this region should be this amount, unless the user has specified a different number of transmit or receive BDs.

This driver can operate only if this memory region is non-cacheable or if the hardware implements bus snooping. The driver cannot maintain cache coherency for the device because the buffers are asynchronously modified by both the driver and the device, and these fields might share the same cache line. Additionally, the chip's dual-ported RAM must be declared as non-cacheable memory where applicable (for example, when attached to a 68040 processor). For more information, see the *Motorola MC68EN360 User's Manual*, *Motorola MPC860 User's Manual*, *Motorola MPC821 User's Manual*

motFecEnd

| | |
|---------------------|---|
| NAME | motFecEnd – END style Motorola FEC Ethernet network interface driver |
| ROUTINES | <i>motFecEndLoad()</i> – initialize the driver and device |
| DESCRIPTION | <p>This module implements a Motorola Fast Ethernet Controller (FEC) network interface driver. The FEC is fully compliant with the IEEE 802.3 10Base-T and 100Base-T specifications. Hardware support of the Media Independent Interface (MII) is built-in in the chip.</p> <p>The FEC establishes a shared memory communication system with the CPU, which is divided into two parts: the Control/Status Registers (CSR), and the buffer descriptors (BD).</p> <p>The CSRs reside in the MPC860T Communication Controller's internal RAM. They are used for mode control and to extract status information of a global nature. For instance, the types of events that should generate an interrupt, or features like the promiscuous mode or the max receive frame length may be set programming some of the CSRs properly. Pointers to both the Transmit Buffer Descriptors ring (TBD) and the Receive Buffer Descriptors ring (RBD) are also stored in the CSRs. The CSRs are located in on-chip RAM and must be accessed using the big-endian mode.</p> <p>The BDs are used to pass data buffers and related buffer information between the hardware and the software. They reside in the host main memory and basically include local status information and a pointer to the actual buffer, again in external memory.</p> <p>This driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below.</p> |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |

EXTERNAL INTERFACE

The driver provides the standard external interface, *motFecEndLoad()*, which takes a string of colon-separated parameters. The parameters should be specified in hexadecimal, optionally preceded by "0x" or a minus sign "-".

The parameter string is parsed using *strtok_r()* and each parameter is converted from a string representation to binary by a call to *strtoul(parameter, NULL, 16)*.

The format of the parameter string is:

```
"motCpmAddr:ivec:bufBase:bufSize:fifoTxBase:fifoRxBase:tbdNum:rbdNum:phyAddr:isoPhyAddr:  
phyDefMode:userFlags"
```

TARGET-SPECIFIC PARAMETERS*motCpmAddr*

Indicates the address at which the host processor presents its internal memory (also known as the dual ported RAM base address). With this address, the driver is able to compute the location of the FEC parameter RAM, and, ultimately, to program the FEC for proper operations.

ivec

This driver configures the host processor to generate hardware interrupts for various events within the device. The interrupt-vector offset parameter is used to connect the driver's ISR to the interrupt through a call to the VxWorks system function *intConnect()*. It is also used to compute the interrupt level (0-7) associated with the FEC interrupt (one of the MPC860T SIU internal interrupt sources). The latter is given as a parameter to *intEnable()*, in order to enable this level interrupt to the PPC core.

bufBase

The Motorola Fast Ethernet Controller is a DMA-type device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the FEC.

This parameter tells the driver that space for the both the TBDs and the RBDs need not be allocated but should be taken from a cache-coherent private memory space provided by the user at the given address. Be aware that memory used for buffers descriptors must be 8-byte aligned and non-cacheable. All the buffer descriptors should fit in the given memory space. If this parameter is "NONE", space for buffer descriptors is obtained by calling *cacheDmaMalloc()* in *motFecEndLoad()*.

bufSize

The memory size parameter specifies the size of the pre-allocated memory region. If *bufBase* is specified as NONE (-1), the driver ignores this parameter. Otherwise, the driver checks the size of the provided memory region is adequate with respect to the given number of Transmit Buffer Descriptors and Receive Buffer Descriptors.

fifoTxBase

Indicate the base location of the transmit FIFO, in internal memory. The user does not need to initialize this parameter, as the related FEC register defaults to a proper value

after reset. The specific reset value is microcode dependent. However, if the user wishes to reserve some RAM for other purposes, he may set this parameter to a different value. This should not be less than the default.

If *fifoTxBase* is specified as NONE (-1), the driver ignores it.

fifoRxBase

Indicate the base location of the receive FIFO, in internal memory. The user does not need to initialize this parameter, as the related FEC register defaults to a proper value after reset. The specific reset value is microcode dependent. However, if the user wishes to reserve some RAM for other purposes, he may set this parameter to a different value. This should not be less than the default.

If *fifoRxBase* is specified as NONE (-1), the driver ignores it.

tbdNum

This parameter specifies the number of transmit buffer descriptors (TBDs). Each buffer descriptor resides in 8 bytes of the processor's external RAM space, and each one points to a 1536-byte buffer again in external RAM. If this parameter is less than a minimum number specified in the macro **MOT_FEC_TBD_MIN**, or if it is "NULL", a default value of 64 is used. This default number is kept deliberately high, since each packet the driver sends may consume more than a single TBD. This parameter should always equal an even number.

rbdNum

This parameter specifies the number of receive buffer descriptors (RBDs). Each buffer descriptor resides in 8 bytes of the processor's external RAM space, and each one points to a 1536-byte buffer again in external RAM. If this parameter is less than a minimum number specified in the macro **MOT_FEC_RBD_MIN**, or if it is "NULL", a default value of 48 is used. This parameter should always equal an even number.

phyAddr

This parameter specifies the logical address of a MII-compliant physical device (PHY) that is to be used as a physical media on the network. Valid addresses are in the range 0-31. There may be more than one device under the control of the same management interface. If this parameter is "NULL", the default physical layer initialization routine will find out the PHY actual address by scanning the whole range. The one with the lowest address will be chosen.

isoPhyAddr

This parameter specifies the logical address of a MII-compliant physical device (PHY) that is to be electrically isolated by the management interface. Valid addresses are in the range 0-31. If this parameter equals 0xff, the default physical layer initialization routine will assume there is no need to isolate any device. However, this parameter will be ignored unless the **MOT_FEC_USR_PHY_ISO** bit in the *userFlags* is set to one.

phyDefMode

This parameter specifies the operating mode that will be set up by the default physical layer initialization routine in case all the attempts made to establish a valid

link failed. If that happens, the first PHY that matches the specified abilities will be chosen to work in that mode, and the physical link will not be tested.

userFlags

This field enables the user to give some degree of customization to the driver, especially as regards the physical layer interface.

MOT_FEC_USR_PHY_NO_AN: the default physical layer initialization routine will exploit the auto-negotiation mechanism as described in the IEEE Std 802.3, to bring a valid link up. According to it, all the link partners on the media will take part to the negotiation process, and the highest priority common denominator technology ability will be chosen. If the user wishes to prevent auto-negotiation from occurring, he may set this bit in the user flags.

MOT_FEC_USR_PHY_TBL: in the auto-negotiation process, PHYs advertise all their technology abilities at the same time, and the result is that the maximum common denominator is used. However, this behaviour may be changed, and the user may affect the order how each subset of PHY's abilities is negotiated. Hence, when the **MOT_FEC_USR_PHY_TBL** bit is set, the default physical layer initialization routine will look at the `motFecPhyAnOrderTbl[]` table and auto-negotiate a subset of abilities at a time, as suggested by the table itself. It is worth noticing here, however, that if the **MOT_FEC_USR_PHY_NO_AN** bit is on, the above table will be ignored.

MOT_FEC_USR_PHY_NO_FD: the PHY may be set to operate in full duplex mode, provided it has this ability, as a result of the negotiation with other link partners. However, in this operating mode, the FEC will ignore the collision detect and carrier sense signals. If the user wishes not to negotiate full duplex mode, he should set the **MOT_FEC_USR_PHY_NO_FD** bit in the user flags.

MOT_FEC_USR_PHY_NO_HD: the PHY may be set to operate in half duplex mode, provided it has this ability, as a result of the negotiation with other link partners. If the user wishes not to negotiate half duplex mode, he should set the **MOT_FEC_USR_PHY_NO_HD** bit in the user flags.

MOT_FEC_USR_PHY_NO_100: the PHY may be set to operate at 100Mbit/s speed, provided it has this ability, as a result of the negotiation with other link partners. If the user wishes not to negotiate 100Mbit/s speed, he should set the **MOT_FEC_USR_PHY_NO_100** bit in the user flags.

MOT_FEC_USR_PHY_NO_10: the PHY may be set to operate at 10Mbit/s speed, provided it has this ability, as a result of the negotiation with other link partners. To not negotiate 10Mbit/s speed, set the **MOT_FEC_USR_PHY_NO_10** bit in the user flags.

MOT_FEC_USR_PHY_ISO: some boards may have different PHYs controlled by the same management interface. In some cases, there may be the need of electrically isolating some of them from the interface itself, in order to guarantee a proper behaviour on the medium layer. If the user wishes to electrically isolate one PHY from the MII interface, he should set the **MOT_FEC_USR_PHY_ISO** bit and provide its logical address in the *isoPhyAddr* field of the load string. The default behaviour is to not isolate any PHY on the board.

MOT_FEC_USR_SER: the user may set the **MOT_FEC_USR_SER** bit to enable the 7-wire interface instead of the MII which is the default.

MOT_FEC_USR_LOOP: when the **MOT_FEC_USR_LOOP** bit is set, the driver will configure the FEC to work in loopback mode, with the TX signal directly connected to the RX. This mode should only be used for testing.

MOT_FEC_USR_HBC: if the **MOT_FEC_USR_HBC** bit is set, the driver configures the FEC to perform heartbeat check following end of transmission and the HB bit in the status field of the TBD will be set if the collision input does not assert within the heartbeat window (also see **_func_motFecHbFail**, below). The user does not normally need to set this bit.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires three external support functions:

sysFecEnetEnable()

STATUS sysFecEnetEnable (UINT32 motCpmAddr);

This routine is expected to handle any target-specific functions needed to enable the FEC. These functions typically include setting the Port D on the 860T-based board so that the MII interface may be used, and also disabling the IRQ7 signal. This routine is expected to return OK on success, or ERROR. The driver calls this routine, once per device, from the *motFecEndLoad()* routine.

sysFecEnetDisable()

STATUS sysFecEnetDisable (UINT32 motCpmAddr);

This routine is expected to perform any target specific functions required to disable the MII interface to the FEC. This involves restoring the default values for all the Port D signals. This routine is expected to return OK on success, or ERROR. The driver calls this routine from the *motFecEndStop()* routine each time a device is disabled.

sysFecEnetAddrGet()

STATUS sysFecEnetAddrGet (UINT32 motCpmAddr, UCHAR * enetAddr);

The driver expects this routine to provide the six-byte Ethernet hardware address that is used by this device. This routine must copy the six-byte address to the space provided by *enetAddr*. This routine is expected to return OK on success, or ERROR. The driver calls this routine, once per device, from the *motFecEndLoad()* routine.

_func_motFecPhyInit

FUNCPTR _func_motFecPhyInit

This driver sets the global variable **_func_motFecPhyInit** to the MII-compliant media initialization routine *motFecPhyInit()*. If the user wishes to exploit a different way to configure the PHY, he may set this variable to his own media initialization routine, typically in *sysHwInit()*.

_func_motFecHbFail

FUNCPTR _func_motFecPhyInit

The FEC may be configured to perform heartbeat check following end of transmission, and to generate an interrupt, when this event occurs. If this is the case, and if the global variable **_func_motFecHbFail** is not NULL, the routine referenced to

by `_func_motFecHbFail` is called, with a pointer to the driver control structure as parameter. Hence, the user may set this variable to his own heart beat check fail routine, where he can take any action he sees appropriate. The default value for the global variable `_func_motFecHbFail` is NULL.

SYSTEM RESOURCE USAGE

If the driver allocates the memory to share with the Ethernet device, it does so by calling the `cacheDmaMalloc()` routine. For the default case of 64 transmit buffers and 48 receive buffers, the total size requested is 912 bytes, and this includes the 16-byte alignment requirement of the device. If a non-cacheable memory region is provided by the user, the size of this region should be this amount, unless the user has specified a different number of transmit or receive BDs.

This driver can operate only if this memory region is non-cacheable or if the hardware implements bus snooping. The driver cannot maintain cache coherency for the device because the BDs are asynchronously modified by both the driver and the device, and these fields might share the same cache line.

Data buffers are instead allocated in the external memory through the regular memory allocation routine (`memalign`), and the related cache lines are then flushed or invalidated as appropriate. The user should not allocate memory for them.

TUNING HINTS

The only adjustable parameters are the number of TBDs and RBDs that will be created at run-time. These parameters are given to the driver when `motFecEndLoad()` is called. There is one RBD associated with each received frame whereas a single transmit packet normally uses more than one TBD. For memory-limited applications, decreasing the number of RBDs may be desirable. Decreasing the number of TBDs below a certain point will provide substantial performance degradation, and is not recommended. An adequate number of loaning buffers are also pre-allocated to provide more buffering before packets are dropped, but this is not configurable.

The relative priority of the netTask and of the other tasks in the system may heavily affect performance of this driver. Usually the best performance is achieved when the netTask priority equals that of the other applications using the driver.

SPECIAL CONSIDERATIONS

Due to the FEC8 errata in the document: "MPC860 Family Device Errata Reference" available at the Motorola web site, the number of receive buffer descriptors (RBD) for the FEC (see `configNet.h`) is kept deliberately high. According to Motorola, this problem was fixed in Rev. B3 of the silicon. In memory-bound applications, when using the above mentioned revision of the MPC860T processor, the user may decrease the number of RBDs to fit his needs.

SEE ALSO

`ifLib`, *MPC860T Fast Ethernet Controller (Supplement to the MPC860 User's Manual)*
Motorola MPC860 User's Manual

mountLib

NAME **mountLib** – Mount protocol library

ROUTINES *mountdInit()* – initialize the mount daemon
nfsExport() – specify a file system to be NFS exported
nfsUnexport() – remove a file system from the list of exported file systems

DESCRIPTION This library implements a mount server to support mounting VxWorks file systems remotely. The mount server is an implementation of version 1 of the mount protocol as defined in RFC 1094. It is closely connected with version 2 of the Network File System Protocol Specification, which in turn is implemented by the library **nfsdLib**.

NOTE The only routines in this library that are normally called by applications are *nfsExport()* and *nfsUnexport()*. The mount daemon is normally initialized indirectly by *nfsdInit()*.

The mount server is initialized by calling *mountdInit()*. Normally, this is done by *nfsdInit()*, although it is possible to call *mountdInit()* directly if the NFS server is not being initialized. Defining **INCLUDE_NFS_SERVER** enables the call to *nfsdInit()* during the boot process, which in turn calls *mountdInit()*, so there is normally no need to call either routine manually. *mountdInit()* spawns one task, **tMountd**, which registers as an RPC service with the portmapper.

Currently, only **dosFsLib** file systems are supported; RT11 file systems cannot be exported. File systems are exported with the *nfsExport()* call.

To export VxWorks file systems via NFS, you need facilities from both this library and from **nfsdLib**. To include both, define the configuration macro **INCLUDE_NFS_SERVER** and rebuild VxWorks.

To initialize a file system to be exported, set **DOS_OPT_EXPORT** in the **DOS_VOL_CONFIG** structure used for initialization. You can do this directly in the *dosFsDevInit()* call, or indirectly with *dosFsDevInitOptionsSet()* or *dosFsMkfsOptionsSet()*.

Example This example illustrates how to initialize and export an existing dosFs file system.

First, initialize the block device containing your file system (identified by *pBlockDevice* below). Then execute the following code on the target:

```
dosFsDevInitOptionsSet (DOS_OPT_EXPORT);        /* make exportable */  
dosFsDevInit ("/export", pBlockDevice, NULL); /* initialize on VxWorks */  
nfsExport ("/export", 0, FALSE, 0);            /* make available remotely */
```

This initializes the DOS file system, and makes it available to all clients to be mounted using the client's NFS mounting command. (On UNIX systems, mounting file systems normally requires root privileges.)

Note that DOS file names are normally limited to 8 characters with a three character extension. You can use an additional initialization option, `DOS_OPT_LONGNAMES`, to enable the VxWorks extension that allows file names up to forty characters long. Replace the `dosFsDevInitOptionsSet()` call in the example above with the following:

```
dosFsMkfsOptionsSet (DOS_OPT_EXPORT | DOS_OPT_LONGNAMES);
```

The variables `dosFsUserId`, `dosFsGroupId`, and `dosFsFileMode` can be set before initialization to specify ownership and permissions as reported over NFS, but they are not required. The defaults appear in the `dosFsLib` manual entry. DOS file systems do not provide for permissions, user IDs, and group IDs on a per-file basis; these variables specify this information for all files on an entire DOS file system.

VxWorks does not normally provide authentication services for NFS requests, and the DOS file system does not provide file permissions. If you need to authenticate incoming requests, see the documentation for `nfsdInit()` and `mountdInit()` for information about authorization hooks.

The following requests are accepted from clients. For details of their use, see Appendix A of RFC 1094, "NFS: Network File System Protocol Specification."

| Procedure Name | Procedure Number |
|-------------------|------------------|
| MOUNTPROC_NULL | 0 |
| MOUNTPROC_MNT | 1 |
| MOUNTPROC_DUMP | 2 |
| MOUNTPROC_UMNT | 3 |
| MOUNTPROC_UMNTALL | 4 |
| MOUNTPROC_EXPORT | 5 |

SEE ALSO `dosFsLib`, `nfsdLib`, RFC 1094

mqPxLib

NAME `mqPxLib` – message queue library (POSIX)

ROUTINES

- `mqPxLibInit()` – initialize the POSIX message queue library
- `mq_open()` – open a message queue (POSIX)
- `mq_receive()` – receive a message from a message queue (POSIX)
- `mq_send()` – send a message to a message queue (POSIX)
- `mq_close()` – close a message queue (POSIX)
- `mq_unlink()` – remove a message queue (POSIX)
- `mq_notify()` – notify a task that a message is available on a queue (POSIX)
- `mq_setattr()` – set message queue attributes (POSIX)
- `mq_getattr()` – get message queue attributes (POSIX)

| | |
|--|---|
| DESCRIPTION | <p>This library implements the message-queue interface defined in the POSIX 1003.1b standard, as an alternative to the VxWorks-specific message queue design in msgQLib. These message queues are accessed through names; each message queue supports multiple sending and receiving tasks.</p> <p>The message queue interface imposes a fixed upper bound on the size of messages that can be sent to a specific message queue. The size is set on an individual queue basis. The value may not be changed dynamically.</p> <p>This interface allows a task be notified asynchronously of the availability of a message on the queue. The purpose of this feature is to let the task to perform other functions and yet still be notified that a message has become available on the queue.</p> |
| MESSAGE QUEUE DESCRIPTOR DELETION | <p>The <i>mq_close()</i> call terminates a message queue descriptor and deallocates any associated memory. When deleting message queue descriptors, take care to avoid interfering with other tasks that are using the same descriptor. Tasks should only close message queue descriptors that the same task has opened successfully.</p> <p>The routines in this library conform to POSIX 1003.1b.</p> |
| INCLUDE FILES | mqqueue.h |
| SEE ALSO | POSIX 1003.1b document, msgQLib , <i>VxWorks Programmer's Guide: Basic OS</i> |

mqPxShow

| | |
|--------------------|--|
| NAME | mqPxShow – POSIX message queue show |
| ROUTINES | <i>mqPxShowInit()</i> – initialize the POSIX message queue show facility |
| DESCRIPTION | This library provides a show routine for POSIX objects. |

msgQLib

| | |
|--|---|
| NAME | msgQLib – message queue library |
| ROUTINES | <i>msgQCreate()</i> – create and initialize a message queue <i>msgQDelete()</i> – delete a message queue <i>msgQSend()</i> – send a message to a message queue <i>msgQReceive()</i> – receive a message from a message queue <i>msgQNumMsgs()</i> – get the number of messages queued to a message queue |
| DESCRIPTION | This library contains routines for creating and using message queues, the primary intertask communication mechanism within a single CPU. Message queues allow a variable number of messages (varying in length) to be queued in first-in-first-out (FIFO) order. Any task or interrupt service routine can send messages to a message queue. Any task can receive messages from a message queue. Multiple tasks can send to and receive from the same message queue. Full-duplex communication between two tasks generally requires two message queues, one for each direction. |
| CREATING AND USING MESSAGE QUEUES | <p>A message queue is created with <i>msgQCreate()</i>. Its parameters specify the maximum number of messages that can be queued to that message queue and the maximum length in bytes of each message. Enough buffer space will be pre-allocated to accommodate the specified number of messages of specified length.</p> <p>A task or interrupt service routine sends a message to a message queue with <i>msgQSend()</i>. If no tasks are waiting for messages on the message queue, the message is simply added to the buffer of messages for that queue. If any tasks are already waiting to receive a message from the message queue, the message is immediately delivered to the first waiting task.</p> <p>A task receives a message from a message queue with <i>msgQReceive()</i>. If any messages are already available in the message queue's buffer, the first message is immediately dequeued and returned to the caller. If no messages are available, the calling task will block and be added to a queue of tasks waiting for messages. This queue of waiting tasks can be ordered either by task priority or FIFO, as specified in an option parameter when the queue is created.</p> |
| TIMEOUTS | Both <i>msgQSend()</i> and <i>msgQReceive()</i> take timeout parameters. When sending a message, if no buffer space is available to queue the message, the timeout specifies how many ticks to wait for space to become available. When receiving a message, the timeout specifies how many ticks to wait if no message is immediately available. The <i>timeout</i> parameter can have the special values NO_WAIT (0) or WAIT_FOREVER (-1). NO_WAIT means the routine should return immediately; WAIT_FOREVER means the routine should never time out. |

URGENT MESSAGES

The *msgQSend()* routine allows the priority of a message to be specified as either normal or urgent, `MSG_PRI_NORMAL` (0) and `MSG_PRI_URGENT` (1), respectively. Normal priority messages are added to the tail of the list of queued messages, while urgent priority messages are added to the head of the list.

INCLUDE FILES `msgQLib.h`

SEE ALSO `pipeDrv`, `msgQSmLib`, *VxWorks Programmer's Guide: Basic OS*

msgQShow

NAME `msgQShow` – message queue show routines

ROUTINES *msgQShowInit()* – initialize the message queue show facility
msgQInfoGet() – get information about a message queue
msgQShow() – show information about a message queue

DESCRIPTION This library provides routines to show message queue statistics, such as the task queuing method, messages queued, receivers blocked, etc.

The routine *msgQshowInit()* links the message queue show facility into the VxWorks system. It is called automatically when the message queue show facility is configured into VxWorks using either of the following methods:

- If you use configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
- If you use the Tornado project facility, select `INCLUDE_MSG_Q_SHOW`.

INCLUDE FILES `msgQLib.h`

SEE ALSO `pipeDrv`, *VxWorks Programmer's Guide: Basic OS*

msgQSmLib

NAME `msgQSmLib` – shared memory message queue library (VxMP Opt.)

ROUTINES *msgQSmCreate()* – create and initialize a shared memory message queue (VxMP Opt.)

DESCRIPTION This library provides the interface to shared memory message queues. Shared memory message queues allow a variable number of messages (varying in length) to be queued in

first-in-first-out order. Any task running on any CPU in the system can send messages to or receive messages from a shared message queue. Tasks can also send to and receive from the same shared message queue. Full-duplex communication between two tasks generally requires two shared message queues, one for each direction.

Shared memory message queues are created with *msgQSmCreate()*. Once created, they can be manipulated using the generic routines for local message queues; for more information on the use of these routines, see the manual entry for **msgQLib**.

MEMORY REQUIREMENTS

The shared memory message queue structure is allocated from a dedicated shared memory partition. This shared memory partition is initialized by the shared memory objects master CPU. The size of this partition is defined by the maximum number of shared message queues, **SM_OBJ_MAX_MSG_Q**.

The message queue buffers are allocated from the shared memory system partition.

RESTRICTIONS Shared memory message queues differ from local message queues in the following ways:

Interrupt Use:

Shared memory message queues may not be used (sent to or received from) at interrupt level.

Deletion:

There is no way to delete a shared memory message queue and free its associated shared memory. Attempts to delete a shared message queue return **ERROR** and set **errno** to **S_smObjLib_NO_OBJECT_DESTROY**.

Queuing Style:

The shared message queue task queuing order specified when a message queue is created must be **FIFO**.

CONFIGURATION Before routines in this library can be called, the shared memory objects facility must be initialized by calling *usrSmObjInit()*, which is found in **src/config/usrSmObj.c**. This is done automatically from the root task, *usrRoot()*, in **usrConfig.c** if the configuration macro **INCLUDE_SM_OBJ** is defined.

AVAILABILITY This module is distributed as a component of the unbundled shared objects memory support option, **VxMP**.

INCLUDE FILES **msgQSmLib.h**, **msgQLib.h**, **smMemLib.h**, **smObjLib.h**

SEE ALSO **msgQLib**, **smObjLib**, **msgQShow**, *usrSmObjInit()*, *VxWorks Programmer's Guide: Shared Memory Objects*

muxLib

| | |
|----------------------|---|
| NAME | muxLib – MUX network interface library |
| ROUTINES | <p><i>muxLibInit()</i> – initialize global state for the MUX <i>muxDevLoad()</i> – load a driver into the MUX <i>muxDevStart()</i> – start a device by calling its start routine <i>muxDevStop()</i> – stop a device by calling its stop routine <i>muxShow()</i> – all configured Enhanced Network Drivers <i>muxBind()</i> – bind a protocol to the MUX given a driver name <i>muxSend()</i> – send a packet out on a network interface <i>muxPollSend()</i> – send a packet on a network interface <i>muxPollReceive()</i> – poll for a packet from a device driver <i>muxIoctl()</i> – send control information to the MUX or to a device <i>muxMCastAddrAdd()</i> – add a multicast address to multicast table for a device <i>muxMCastAddrDel()</i> – delete a multicast address from a device’s multicast table <i>muxMCastAddrGet()</i> – get the multicast address table from the MUX/Driver <i>muxUnbind()</i> – detach a protocol from the specified driver <i>muxDevUnload()</i> – remove a driver from the MUX <i>muxAddressForm()</i> – form an address into a packet <i>muxPacketDataGet()</i> – return the data from a packet <i>muxPacketAddrGet()</i> – get addressing information from a packet <i>endFindByName()</i> – find a device using its string name <i>muxDevExists()</i> – tests whether a device is already loaded into the MUX <i>muxAddrResFuncAdd()</i> – add an address resolution function <i>muxAddrResFuncGet()</i> – get the address resolution function for ifType/protocol <i>muxAddrResFuncDel()</i> – delete an address resolution function</p> |
| DESCRIPTION | <p>This library provides the routines that define the MUX interface, a facility that handles communication between the data link layer and the network protocol layer. Using the MUX, the VxWorks network stack has decoupled the data link and network layers. Thus, drivers and protocols no longer need knowledge of each other’s internals. As a result, the network driver and protocol are nearly independent of each another. This independence makes it much easier to add a new drivers or protocols. For example, if you add a new END, all existing MUX-based protocols can use the new driver. Likewise, if you add a new MUX-based protocol, any existing END can use the MUX to access the new protocol.</p> |
| INCLUDE FILES | errno.h, lstLib.h, logLib.h, string.h, m2Lib.h, bufLib.h, if.h, end.h, muxLib.h |
| SEE ALSO | <i>Network Protocol Toolkit User’s Guide</i> |

ncr710CommLib

| | |
|--------------------|---|
| NAME | ncr710CommLib – common library for ncr710Lib.c and ncr710Lib2.c |
| ROUTINES | <i>ncr710SingleStep()</i> – perform a single-step <i>ncr710StepEnable()</i> – enable/disable script single-step |
| DESCRIPTION | Contains ncr710Lib and ncr710Lib2 common driver interfaces which can be called from user code. |
| SEE ALSO | ncr710Lib.c , ncr710Lib2.c , <i>NCR 53C710 SCSI I/O Processor Programming Guide</i> , <i>VxWorks Programmer's Guide: I/O System</i> |

ncr710Lib

| | |
|-------------------------------|--|
| NAME | ncr710Lib – NCR 53C710 SCSI I/O Processor (SIOP) library (SCSI-1) |
| ROUTINES | <i>ncr710CtrlCreate()</i> – create a control structure for an NCR 53C710 SIOP <i>ncr710CtrlInit()</i> – initialize a control structure for an NCR 53C710 SIOP <i>ncr710SetHwRegister()</i> – set hardware-dependent registers for the NCR 53C710 SIOP <i>ncr710Show()</i> – display the values of all readable NCR 53C710 SIOP registers |
| DESCRIPTION | This is the I/O driver for the NCR 53C710 SCSI I/O Processor (SIOP). It is designed to work with scsi1Lib . It also runs in conjunction with a script program for the NCR 53C710 chip. This script uses the NCR 53C710 DMA function for data transfers. This driver supports cache functions through cacheLib . |
| USER-CALLABLE ROUTINES | Most of the routines in this driver are accessible only through the I/O system. Three routines, however, must be called directly: <i>ncr710CtrlCreate()</i> to create a controller structure, and <i>ncr710CtrlInit()</i> to initialize it. The NCR 53C710 hardware registers need to be configured according to the hardware implementation. If the default configuration is not proper, the routine <i>ncr710SetHwRegister()</i> should be used to properly configure the registers. |
| INCLUDE FILES | ncr710.h , ncr710_1.h , ncr710Script.h , ncr710Script1.h |
| SEE ALSO | scsiLib , scsi1Lib , cacheLib , <i>NCR 53C710 SCSI I/O Processor Programming Guide</i> , <i>VxWorks Programmer's Guide: I/O System</i> |

ncr710Lib2

| | |
|-------------------------------|---|
| NAME | ncr710Lib2 – NCR 53C710 SCSI I/O Processor (SIOP) library (SCSI-2) |
| ROUTINES | <i>ncr710CtrlCreateScsi2()</i> – create a control structure for the NCR 53C710 SIOP <i>ncr710CtrlInitScsi2()</i> – initialize a control structure for the NCR 53C710 SIOP <i>ncr710SetHwRegisterScsi2()</i> – set hardware-dependent registers for the NCR 53C710 <i>ncr710ShowScsi2()</i> – display the values of all readable NCR 53C710 SIOP registers |
| DESCRIPTION | This is the I/O driver for the NCR 53C710 SCSI I/O Processor (SIOP). It is designed to work with scsi2Lib . This driver runs in conjunction with a script program for the NCR 53C710 chip. The script uses the NCR 53C710 DMA function for data transfers. This driver supports cache functions through cacheLib . |
| USER-CALLABLE ROUTINES | Most of the routines in this driver are accessible only through the I/O system. Three routines, however, must be called directly. <i>ncr710CtrlCreateScsi2()</i> creates a controller structure and <i>ncr710CtrlInitScsi2()</i> initializes it. The NCR 53C710 hardware registers need to be configured according to the hardware implementation. If the default configuration is not correct, the routine <i>ncr710SetHwRegisterScsi2()</i> must be used to properly configure the registers. |
| INCLUDE FILES | ncr710.h , ncr710_2.h , ncr710Script.h , ncr710Script2.h |
| SEE ALSO | scsiLib , scsi2Lib , cacheLib , <i>VxWorks Programmer's Guide: I/O System</i> |

ncr810Lib

| | |
|-------------------------------|--|
| NAME | ncr810Lib – NCR 53C8xx PCI SCSI I/O Processor (SIOP) library (SCSI-2) |
| ROUTINES | <i>ncr810CtrlCreate()</i> – create a control structure for the NCR 53C8xx SIOP <i>ncr810CtrlInit()</i> – initialize a control structure for the NCR 53C8xx SIOP <i>ncr810SetHwRegister()</i> – set hardware-dependent registers for the NCR 53C8xx SIOP <i>ncr810Show()</i> – display values of all readable NCR 53C8xx SIOP registers |
| DESCRIPTION | This is the I/O driver for the NCR 53C8xx PCI SCSI I/O Processors (SIOP), supporting the NCR 53C810 and the NCR 53C825 SCSI controllers. It is designed to work with scsiLib and scsi2Lib . This driver runs in conjunction with a script program for the NCR 53C8xx controllers. These scripts use DMA transfers for all data, messages, and status. This driver supports cache functions through cacheLib . |
| USER-CALLABLE ROUTINES | Most of the routines in this driver are accessible only through the I/O system. Three routines, however, must be called directly. <i>ncr810CtrlCreate()</i> creates a controller structure and <i>ncr810CtrlInit()</i> initializes it. The NCR 53C8xx hardware registers need to be configured according to the hardware implementation. If the default configuration is not correct, the routine <i>ncr810SetHwRegister()</i> must be used to properly configure the registers. |
| PCI MEMORY ADDRESSING | The global variable <code>ncr810PciMemOffset</code> was created to provide the BSP with a means of changing the <code>VIRT_TO_PHYS</code> mapping without changing the functions in the <code>cacheFuncs</code> structures. In generating physical addresses for DMA on the PCI bus, local addresses are passed through the function <code>CACHE_DMA_VIRT_TO_PHYS</code> and then the value of <code>ncr810PciMemOffset</code> is added. For backward compatibility, the initial value of <code>ncr810PciMemOffset</code> comes from the macro <code>PCI_TO_MEM_OFFSET</code> defined in ncr810.h . I/O MACROS All device access for input and output is done via macros which can be customized for each BSP. These routines are <code>NCR810_IN_BYTE</code> , <code>NCR810_OUT_BYTE</code> , <code>NCR810_IN_16</code> , <code>NCR810_OUT_16</code> , <code>NCR810_IN_32</code> and <code>NCR810_OUT_32</code> . By default, these are defined as generic memory references. |
| INCLUDE FILES | ncr810.h , ncr810Script.h and scsiLib.h |
| SEE ALSO | scsiLib , scsi2Lib , cacheLib , <i>SYM53C825 PCI-SCSI I/O Processor Data Manual</i> , <i>SYM53C810 PCI-SCSI I/O Processor Data Manual</i> , <i>NCR 53C8XX Family PCI-SCSI I/O Processors Programming Guide</i> , <i>VxWorks Programmer's Guide: I/O System</i> |

ncr5390Lib

| | |
|----------------------|---|
| NAME | ncr5390Lib – NCR5390 SCSI-Bus Interface Controller library (SBIC) |
| ROUTINES | <i>ncr5390CtrlInit()</i> – initialize the user-specified fields in an ASC structure <i>ncr5390Show()</i> – display the values of all readable NCR5390 chip registers |
| DESCRIPTION | <p>This library contains the main interface routines to the SCSI-Bus Interface Controllers (SBIC). These routines simply switch the calls to the SCSI-1 or SCSI-2 drivers, implemented in ncr5390Lib1.c or ncr5390Lib2.c as configured by the Board Support Package (BSP).</p> <p>In order to configure the SCSI-1 driver, which depends upon scsi1Lib, the <i>ncr5390CtrlCreate()</i> routine, defined in ncr5390Lib1, must be invoked. Similarly <i>ncr5390CtrlCreateScsi2()</i>, defined in ncr5390Lib2 and dependent on scsi2Lib, must be called to configure and initialize the SCSI-2 driver.</p> |
| INCLUDE FILES | ncr5390.h , ncr5390_1.h , ncr5390_2.h |

ncr5390Lib1

| | |
|-------------------------------|---|
| NAME | ncr5390Lib1 – NCR 53C90 Advanced SCSI Controller (ASC) library (SCSI-1) |
| ROUTINES | <i>ncr5390CtrlCreate()</i> – create a control structure for an NCR 53C90 ASC |
| DESCRIPTION | This is the I/O driver for the NCR 53C90 Advanced SCSI Controller (ASC). It is designed to work in conjunction with scsiLib . |
| USER-CALLABLE ROUTINES | <p>Most of the routines in this driver are accessible only through the I/O system. The only exception in this portion of the driver is the <i>ncr5390CtrlCreate()</i> which creates a controller structure.</p> |
| INCLUDE FILES | ncr5390.h |
| SEE ALSO | scsiLib , <i>NCR 53C90A, 53C90B Advanced SCSI Controller</i> , <i>VxWorks Programmer's Guide: I/O System</i> |

ncr5390Lib2

| | |
|-------------------------------|---|
| NAME | ncr5390Lib2 – NCR 53C90 Advanced SCSI Controller (ASC) library (SCSI-2) |
| ROUTINES | <i>ncr5390CtrlCreateScsi2()</i> – create a control structure for an NCR 53C90 ASC |
| DESCRIPTION | This is the I/O driver for the NCR 53C90 Advanced SCSI Controller (ASC). It is designed to work in conjunction with scsiLib . |
| USER-CALLABLE ROUTINES | Most of the routines in this driver are accessible only through the I/O system. The only exception in this portion of the driver is the <i>ncr5390CtrlCreateScsi2()</i> which creates a controller structure. |
| INCLUDE FILES | ncr5390.h |
| SEE ALSO | ncr5390Lib2 , scsiLib , <i>NCR 53C90A, 53C90B Advanced SCSI Controller, VxWorks Programmer's Guide: I/O System</i> |

ne2000End

| | |
|-----------------------------------|--|
| NAME | ne2000End – NE2000 END network interface driver |
| ROUTINES | <i>ne2000EndLoad()</i> – initialize the driver and device <i>ne2000Parse()</i> – parse the init string |
| DESCRIPTION | This module implements the NE2000 Ethernet network interface driver. |
| EXTERNAL INTERFACE | The only external interface is the <i>ne2000EndLoad()</i> routine, which expects the <i>initString</i> parameter as input. This parameter passes in a colon-delimited string of the format: <i>unit:adrs:vecNum:intLvl:byteAccess:usePromEnetAddr:offset</i> The <i>ne2000EndLoad()</i> function uses <i>strtok()</i> to parse the string. |
| TARGET-SPECIFIC PARAMETERS | <i>unit</i> A convenient holdover from the former model. This parameter is used only in the string name for the driver. |

adrs

Tells the driver where to find the ne2000.

vecNum

Configures the ne2000 device to generate hardware interrupts for various events within the device. Thus, it contains an interrupt handler routine. The driver calls **sysIntConnect()** to connect its interrupt handler to the interrupt vector generated as a result of the ne2000 interrupt.

intLvl

This parameter is passed to an external support routine, **sysLanIntEnable()**, which is described below in "External Support Requirements." This routine is called during as part of driver's initialization. It handles any board-specific operations required to allow the servicing of a ne2000 interrupt on targets that use additional interrupt controller devices to help organize and service the various interrupt sources. This parameter makes it possible for this driver to avoid all board-specific knowledge of such devices.

byteAccess

Tells the driver the NE2000 is jumpered to operate in 8-bit mode. Requires that **SYS_IN_WORD_STRING()** and **SYS_OUT_WORD_STRING()** be written to properly access the device in this mode.

usePromEnetAddr

Attempt to get the ethernet address for the device from the on-chip (board) PROM attached to the NE2000. Will fall back to using the BSP-supplied ethernet address if this parameter is 0 or if unable to read the ethernet address.

offset

Specifies the memory alignment offset.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires several external support functions, defined as macros:

```
SYS_INT_CONNECT(pDrvCtrl, routine, arg)
SYS_INT_DISCONNECT (pDrvCtrl, routine, arg)
SYS_INT_ENABLE(pDrvCtrl)
SYS_IN_CHAR(pDrvCtrl, reg, pData)
SYS_OUT_CHAR(pDrvCtrl, reg, pData)
SYS_IN_WORD_STRING(pDrvCtrl, reg, pData)
SYS_OUT_WORD_STRING(pDrvCtrl, reg, pData)
```

These macros allow the driver to be customized for BSPs that use special versions of these routines.

The macro **SYS_INT_CONNECT** is used to connect the interrupt handler to the appropriate vector. By default it is the routine **intConnect()**.

The macro **SYS_INT_DISCONNECT** is used to disconnect the interrupt handler prior to unloading the module. By default this is a dummy routine that returns OK.

The macro `SYS_INT_ENABLE` is used to enable the interrupt level for the end device. It is called once during initialization. By default this is the routine `sysLanIntEnable()`, defined in the module `sysLib.o`.

The macro `SYS_ENET_ADDR_GET` is used to get the ethernet address (MAC) for the device. The single argument to this routine is the `END_DEVICE` pointer. By default this routine copies the ethernet address stored in the global variable `ne2000EndEnetAddr` into the `END_DEVICE` structure.

The macros `SYS_IN_CHAR`, `SYS_OUT_CHAR`, `SYS_IN_WORD_STRING` and `SYS_OUT_WORD_STRING` are used for accessing the ne2000 device. The default macros map these operations onto `sysInByte()`, `sysOutByte()`, `sysInWordString()` and `sysOutWordString()`.

INCLUDES `end.h` `endLib.h` `etherMultiLib.h`

SEE ALSO `muxLib`, `endLib` *Writing and Enhanced Network Driver*

nec765Fd

NAME `nec765Fd` – NEC 765 floppy disk device driver

ROUTINES `fdDrv()` – initialize the floppy disk driver
`fdDevCreate()` – create a device for a floppy disk
`fdRawio()` – provide raw I/O access

DESCRIPTION This is the driver for the NEC 765 Floppy Chip used on the PC 386/486.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. However, two routines must be called directly: `fdDrv()` to initialize the driver, and `fdDevCreate()` to create devices. Before the driver can be used, it must be initialized by calling `fdDrv()`. This routine should be called exactly once, before any reads, writes, or calls to `fdDevCreate()`. Normally, it is called from `usrRoot()` in `usrConfig.c`.

The routine `fdRawio()` allows physical I/O access. Its first argument is a drive number, 0 to 3; the second argument is a type of diskette; the third argument is a pointer to the `FD_RAW` structure, which is defined in `nec765Fd.h`.

Interleaving is not supported when the driver formats.

Two types of diskettes are currently supported: 3.5" 2HD 1.44MB and 5.25" 2HD 1.2MB. You can add additional diskette types to the `fdTypes[]` table in `sysLib.c`.

SEE ALSO *VxWorks Programmer's Guide: I/O System*

netBufLib

NAME netBufLib – network buffer library

ROUTINES

- netBufLibInit()* – initialize **netBufLib**
- netPoolInit()* – initialize a **netBufLib**-managed memory pool
- netPoolDelete()* – delete a memory pool
- netMblkFree()* – free an **mBlk** back to its memory pool
- netCBlkFree()* – free a **cBlk**-cluster construct back to the memory pool
- netClFree()* – free a cluster back to the memory pool
- netMblkClFree()* – free an **mBlk-cBlk**-cluster construct
- netMblkClChainFree()* – free a chain of **mBlk-cBlk**-cluster constructs
- netMblkGet()* – get an **mBlk**
- netCBlkGet()* – get a **cBlk**
- netClusterGet()* – get a cluster from the specified cluster pool
- netMblkClGet()* – get a **cBlk**-cluster and join it to the specified **mBlk**
- netTupleGet()* – get an **mBlk-cBlk**-cluster
- netCBlkJoin()* – join a cluster to a **cBlk** structure
- netMblkClJoin()* – join an **mBlk** to a **cBlk**-cluster construct
- netClPoolIdGet()* – return a **CL_POOL_ID** for a specified buffer size
- netMblkToBufCopy()* – copy data from an **mBlk** to a buffer
- netMblkDup()* – duplicate an **mBlk**
- netMblkChainDup()* – duplicate an **mBlk** chain

DESCRIPTION This library contains routines that you can use to organize and maintain a memory pool that consists of pools of **mBlk** structures, pools of **cBlk** structures, and pools of clusters. The **mBlk** and **cBlk** structures are used to manage the clusters. The clusters are containers for the data described by the **mBlk** and **cBlk** structures.

These structures and the various routines of this library constitute a buffering API that has been designed to meet the needs both of network protocols and network device drivers.

The **mBlk** structure is the primary vehicle for passing data between a network driver and a protocol. However, the **mBlk** structure must first be properly joined with a **cBlk** structure that was previously joined with a cluster. Thus, the actual vehicle for passing data is not merely an **mBlk** structure but an **mBlk-cBlk**-cluster construct.

To include **netBufLib** in VxWorks, define **INCLUDE_NETWORK** in **configAll.h**. This also automatically configures VxWorks to call *netBufLibInit()*.

INCLUDE FILES netBufLib.h

netDrv

- NAME** *netDrv* – network remote file I/O driver
- ROUTINES** *netDrv()* – install the network remote file driver
netDevCreate() – create a remote file device
- DESCRIPTION** This driver provides facilities for accessing files transparently over the network via FTP or RSH. By creating a network device with *netDevCreate()*, files on a remote UNIX machine may be accessed as if they were local.
- When a remote file is opened, the entire file is copied over the network to a local buffer. When a remote file is created, an empty local buffer is opened. Any reads, writes, or *ioctl()* calls are performed on the local copy of the file. If the file was opened with the flags **O_WRONLY** or **O_RDWR** and modified, the local copy is sent back over the network to the UNIX machine when the file is closed.
- Note that this copying of the entire file back and forth can make **netDrv** devices awkward to use. A preferable mechanism is NFS as provided by *nfsDrv*.
- USER-CALLABLE ROUTINES**
- Most of the routines in this driver are accessible only through the I/O system. However, two routines must be called directly: *netDrv()* to initialize the driver and *netDevCreate()* to create devices.
- FILE OPERATIONS** This driver supports the creation, deletion, opening, reading, writing, and appending of files. The renaming of files is not supported.
- INITIALIZATION** Before using the driver, it must be initialized by calling the routine *netDrv()*. This routine should be called only once, before any reads, writes, or *netDevCreate()* calls. Initialization is performed automatically when the configuration macro **INCLUDE_NETWORK** is defined.
- CREATING NETWORK DEVICES**
- To access files on a remote host, a network device must be created by calling *netDevCreate()*. The arguments to *netDevCreate()* are the name of the device, the name of the host the device will access, and the remote file access protocol to be used -- RSH or FTP. By convention, a network device name is the remote machine name followed by a colon ":". For example, for a UNIX host on the network "wrs", files can be accessed by creating a device called "wrs:". For more information, see the manual entry for *netDevCreate()*.
- IOCTL FUNCTIONS** The network driver responds to the following *ioctl()* functions:

netDrv**FIOGETNAME**

Gets the file name of the file descriptor *fd* and copies it to the buffer specified by *nameBuf*:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

FIONREAD

Copies to *nBytesUnread* the number of bytes remaining in the file specified by *fd*:

```
status = ioctl (fd, FIONREAD, &nBytesUnread);
```

FIOSEEK

Sets the current byte offset in the file to the position specified by *newOffset*. If the seek goes beyond the end-of-file, the file grows. The end-of-file pointer changes to the new position, and the new space is filled with zeroes:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

FIOWHERE

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

FIOFSTATGET

Gets file status information. The argument *statStruct* is a pointer to a stat structure that is filled with data describing the specified file. Normally, the *stat()* or *fstat()* routine is used to obtain file information, rather than using the **FIOFSTATGET** function directly. **netDrv** only fills in three fields of the stat structure: *st_dev*, *st_mode*, and *st_size*. *st_mode* is always filled with **S_IFREG**.

```
struct stat statStruct;
fd = open ("file", O_RDONLY);
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

LIMITATIONS

The **netDrv** implementation strategy implies that directories cannot always be distinguished from plain files. Thus, *opendir()* does not work for directories mounted on **netDrv** devices, and *ll()* does not flag subdirectories with the label "DIR" in listings from **netDrv** devices.

When the access method is FTP, operations can only be done on files that the FTP server allows to download. In particular it is not possible to stat a directory, doing so will result in "*dirname*: not a plain file" error.

INCLUDE FILES

netDrv.h

SEE ALSO

remLib, **netLib**, **sockLib**, *hostAdd()*, *VxWorks Programmer's Guide: Network*

netLib

| | |
|----------------------|---|
| NAME | netLib – network interface library |
| ROUTINES | <i>netLibInit()</i> – initialize the network package <i>netTask()</i> – network task entry point |
| DESCRIPTION | <p>This library contains the network task that runs low-level network interface routines in a task context. The network task executes and removes routines that were added to the job queue. This facility is used by network interfaces in order to have interrupt-level processing at task level.</p> <p>The routine <i>netLibInit()</i> initializes the network and spawns the network task <i>netTask()</i>. This is done automatically when the configuration macro <code>INCLUDE_NETWORK</code> is defined.</p> <p>The routine <i>netHelp()</i> in usrLib displays a summary of the network facilities available from the VxWorks shell.</p> |
| INCLUDE FILES | netLib.h |
| SEE ALSO | routeLib , hostLib , netDrv , <i>netHelp()</i> , <i>VxWorks Programmer's Guide: Network</i> |

netShow

| | |
|-----------------|---|
| NAME | netShow – network information display routines |
| ROUTINES | <i>ifShow()</i> – display the attached network interfaces <i>inetstatShow()</i> – display all active connections for Internet protocol sockets <i>ipstatShow()</i> – display IP statistics <i>netPoolShow()</i> – show pool statistics <i>netStackDataPoolShow()</i> – show network stack data pool statistics <i>netStackSysPoolShow()</i> – show network stack system pool statistics <i>mbufShow()</i> – report mbuf statistics <i>netShowInit()</i> – initialize network show routines <i>arpShow()</i> – display entries in the system ARP table <i>arptabShow()</i> – display the known ARP entries <i>routestatShow()</i> – display routing statistics <i>routeShow()</i> – display host and network routing tables <i>hostShow()</i> – display the host table <i>mRouteShow()</i> – print the entries of the routing table |

DESCRIPTION This library provides routines to show various network-related statistics, such as configuration parameters for network interfaces, protocol statistics, socket statistics, and so on.

Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:

- *Internetworking with TCP/IP Volume III*, by Douglas Comer and David Stevens
- *UNIX Network Programming*, by Richard Stevens
- *The Design and Implementation of the 4.3 BSD UNIX Operating System*, by Leffler, McKusick, Karels and Quarterman

The *netShowInit()* routine links the network show facility into the VxWorks system. This is performed automatically if `INCLUDE_NET_SHOW` is defined in `configAll.h`.

SEE ALSO `ifLib`, `icmpShow`, `igmpShow`, `tcpShow`, `udpShow`, *VxWorks Programmer's Guide: Network*

nfsdLib

NAME `nfsdLib` – Network File System (NFS) server library

ROUTINES *nfsdInit()* – initialize the NFS server
nfsdStatusGet() – get the status of the NFS server
nfsdStatusShow() – show the status of the NFS server

DESCRIPTION This library is an implementation of version 2 of the Network File System Protocol Specification as defined in RFC 1094. It is closely connected with version 1 of the mount protocol, also defined in RFC 1094 and implemented in turn by `mountLib`.

The NFS server is initialized by calling *nfsdInit()*. This is done automatically at boot time if the configuration macro `INCLUDE_NFS_SERVER` is defined.

Currently, only `dosFsLib` file systems are supported; RT11 file systems cannot be exported. File systems are exported with the *nfsExport()* call.

To create and export a file system, define the configuration macro `INCLUDE_NFS_SERVER` and rebuild VxWorks.

To export VxWorks file systems via NFS, you need facilities from both this library and from `mountLib`. To include both, define `INCLUDE_NFS_SERVER` and rebuild VxWorks.

Use the `mountLib` routine *nfsExport()* to export file systems. For an example, see the manual page for `mountLib`.

VxWorks does not normally provide authentication services for NFS requests, and the DOS file system does not provide file permissions. If you need to authenticate incoming

requests, see the documentation for *nfsdInit()* and *mountdInit()* for information about authorization hooks.

The following requests are accepted from clients. For details of their use, see RFC 1094, "NFS: Network File System Protocol Specification."

| Procedure Name | Procedure Number |
|------------------|------------------|
| NFSPROC_NULL | 0 |
| NFSPROC_GETATTR | 1 |
| NFSPROC_SETATTR | 2 |
| NFSPROC_ROOT | 3 |
| NFSPROC_LOOKUP | 4 |
| NFSPROC_READLINK | 5 |
| NFSPROC_READ | 6 |
| NFSPROC_WRITE | 8 |
| NFSPROC_CREATE | 9 |
| NFSPROC_REMOVE | 10 |
| NFSPROC_RENAME | 11 |
| NFSPROC_LINK | 12 |
| NFSPROC_SYMLINK | 13 |
| NFSPROC_MKDIR | 14 |
| NFSPROC_RMDIR | 15 |
| NFSPROC_READDIR | 16 |
| NFSPROC_STATFS | 17 |

AUTHENTICATION AND PERMISSIONS

Currently, no authentication is done on NFS requests. *nfsdInit()* describes the authentication hooks that can be added should authentication be necessary.

Note that the DOS file system does not provide information about ownership or permissions on individual files. Before initializing a dosFs file system, three global variables--*dosFsUserId*, *dosFsGroupId*, and *dosFsFileMode*--can be set to define the user ID, group ID, and permissions byte for all files in all dosFs volumes initialized after setting these variables. To arrange for different dosFs volumes to use different user and group ID numbers, reset these variables before each volume is initialized. See the manual entry for *dosFsLib* for more information.

TASKS

Several NFS tasks are created by *nfsdInit()*. They are:

tMountd

The mount daemon, which handles all incoming mount requests. This daemon is created by *mountdInit()*, which is automatically called from *nfsdInit()*.

tNfsd

The NFS daemon, which queues all incoming NFS requests.

tNfsdX

The NFS request handlers, which dequeues and processes all incoming NFS requests.

Performance of the NFS file system can be improved by increasing the number of servers specified in the *nfsdInit()* call, if there are several different dosFs volumes exported from the same target system. The *spy()* utility can be called to determine whether this is useful for a particular configuration.

SEE ALSO **nfsdLib**

nfsDrv

NAME **nfsDrv** – Network File System (NFS) I/O driver

ROUTINES *nfsDrv()* – install the NFS driver
nfsDrvNumGet() – return the IO system driver number for the nfs driver
nfsMount() – mount an NFS file system
nfsMountAll() – mount all file systems exported by a specified host
nfsDevShow() – display the mounted NFS devices
nfsUnmount() – unmount an NFS device
nfsDevListGet() – create list of all the NFS devices in the system
nfsDevInfoGet() – read configuration information from the requested NFS device

DESCRIPTION This driver provides facilities for accessing files transparently over the network via NFS (Network File System). By creating a network device with *nfsMount()*, files on a remote NFS system (such as a UNIX system) can be handled as if they were local.

USER-CALLABLE ROUTINES

The *nfsDrv()* routine initializes the driver. The *nfsMount()* and *nfsUnmount()* routines mount and unmount file systems. The *nfsMountAll()* routine mounts all file systems exported by a specified host.

INITIALIZATION Before using the network driver, it must be initialized by calling *nfsDrv()*. This routine must be called before any reads, writes, or other NFS calls. This is done automatically when the configuration macro `INCLUDE_NFS` is defined.

CREATING NFS DEVICES

To access a remote file system, an NFS device must be created by calling *nfsMount()*. For example, to create the device `/myd0/` for the file system `/d0/` on the host `wrs`, call:

```
nfsMount ("wrs", "/d0/", "/myd0/");
```

The file `/d0/dog` on the host `wrs` can now be accessed as `/myd0/dog`.

If the third parameter to *nfsMount()* is NULL, VxWorks creates a device with the same name as the file system. For example, the call:

```
nfsMount ("wrs", "/d0/", NULL);
```

or from the shell:

```
nfsMount "wrs", "/d0/"
```

creates the device */d0/*. The file */d0/dog* is accessed by the same name, */d0/dog*.

Before mounting a file system, the host must already have been created with *hostAdd()*. The routine *nfsDevShow()* displays the mounted NFS devices.

IOCTL FUNCTIONS The NFS driver responds to the following *ioctl()* functions:

FIOGETNAME

Gets the file name of *fd* and copies it to the buffer referenced by *nameBuf*:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

FIONREAD

Copies to *nBytesUnread* the number of bytes remaining in the file specified by *fd*:

```
status = ioctl (fd, FIONREAD, &nBytesUnread);
```

FIOSEEK

Sets the current byte offset in the file to the position specified by *newOffset*. If the seek goes beyond the end-of-file, the file grows. The end-of-file pointer gets moved to the new position, and the new space is filled with zeros:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

FIOSYNC

Flush data to the remote NFS file. It takes no additional argument:

```
status = ioctl (fd, FIOSYNC, 0);
```

FIOWHERE

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

FIOREADDIR

Reads the next directory entry. The argument *dirStruct* is a pointer to a directory descriptor of type DIR. Normally, the *readdir()* routine is used to read a directory, rather than using the FIOREADDIR function directly. See the manual entry for *dirLib*:

```
DIR dirStruct;
fd = open ("directory", O_RDONLY);
status = ioctl (fd, FIOREADDIR, &dirStruct);
```

FIOFSTATGET

Gets file status information (directory entry data). The argument *statStruct* is a

pointer to a stat structure that is filled with data describing the specified file. Normally, the *stat()* or *fstat()* routine is used to obtain file information, rather than using the FIOFSTATGET function directly. See the manual entry for **dirLib**:

```
struct stat statStruct;
fd = open ("file", O_RDONLY);
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

FIOFSTATFSGET

Gets the file system parameters for an open file descriptor. The argument *statfsStruct* is a pointer to a statfs structure that is filled with data describing the underlying filesystem. Normally, the *stat()* or *fstat()* routine is used to obtain file information, rather than using the FIOFSTATFSGET function directly. See the manual entry for **dirLib**:

```
statfs statfsStruct;
fd = open ("directory", O_RDONLY);
status = ioctl (fd, FIOFSTATFSGET, &statfsStruct);
```

- DEFICIENCIES** There is only one client handle/cache per task. Performance is poor if a task is accessing two or more NFS files.
- Changing *nfsCacheSize* after a file is open could cause adverse effects. However, changing it before opening any NFS file descriptors should not pose a problem.
- INCLUDE FILES** *nfsDrv.h*, *ioLib.h*, *dirent.h*
- SEE ALSO** *dirLib*, *nfsLib*, *hostAdd()*, *ioctl()*, *VxWorks Programmer's Guide: Network*

nfsLib

- NAME** *nfsLib* – Network File System (NFS) library
- ROUTINES** *nfsHelp()* – display the NFS help menu
nfsExportShow() – display the exported file systems of a remote host
nfsAuthUnixPrompt() – modify the NFS UNIX authentication parameters
nfsAuthUnixShow() – display the NFS UNIX authentication parameters
nfsAuthUnixSet() – set the NFS UNIX authentication parameters
nfsAuthUnixGet() – get the NFS UNIX authentication parameters
nfsIdSet() – set the ID number of the NFS UNIX authentication parameters
- DESCRIPTION** This library provides the client side of services for NFS (Network File System) devices. Most routines in this library should not be called by users, but rather by device drivers.

The driver is responsible for keeping track of file pointers, mounted disks, and cached buffers. This library uses Remote Procedure Calls (RPC) to make the NFS calls.

VxWorks is delivered with NFS disabled. The configuration macro for NFS is **INCLUDE_NFS**.

In the same file, **NFS_USER_ID** and **NFS_GROUP_ID** should be defined to set the default user ID and group ID at system start-up. For information about creating NFS devices, see the *VxWorks Programmer's Guide: Network*.

Normal use of NFS requires no more than 2000 bytes of stack.

NFS USER IDENTIFICATION

NFS is built on top of RPC and uses a type of RPC authentication known as **AUTH_UNIX**, which is passed onto the NFS server with every NFS request. **AUTH_UNIX** is a structure that contains necessary information for NFS, including the user ID number and a list of group IDs to which the user belongs. On UNIX systems, a user ID is specified in the file */etc/passwd*. The list of groups to which a user belongs is specified in the file */etc/group*.

To change the default authentication parameters, use *nfsAuthUnixPrompt()*. To change just the **AUTH_UNIX** ID, use *nfsIdSet()*. Usually, only the user ID needs to be changed to indicate a new NFS user.

INCLUDE FILES **nfsLib.h**

SEE ALSO **rpcLib, ioLib, nfsDrv**, *VxWorks Programmer's Guide: Network*

nicEvbEnd

NAME **nicEvbEnd** – National Semiconductor ST-NIC Chip network interface driver

ROUTINES *nicEndLoad()* – initialize the driver and device
nicEvbInitParse() – parse the initialization string

DESCRIPTION This module implements the National Semiconductor 83902A ST-NIC Ethernet network interface driver.

This driver is non-generic and is for use on the IBM EVB403 board. The driver must be given several target-specific parameters. These parameters, and the mechanisms used to communicate them to the driver, are detailed below.

BOARD LAYOUT This device is on-board. No jumpering diagram is necessary.

EXTERNAL INTERFACE

The only external interface is the *nicEvbEndLoad()* routine, which expects the *initString* parameter as input. This parameter passes in a colon-delimited string of the format:

unit:nic_addr:int_vector:int_level

The *nicEvbEndLoad()* function uses *strtok()* to parse the string.

TARGET-SPECIFIC PARAMETERS

unit

A convenient holdover from the former model. This parameter is used only in the string name for the driver.

nic_addr

Base address for NIC chip

int_vector

Configures the NIC device to generate hardware interrupts for various events within the device. Thus, it contains an interrupt handler routine. The driver calls *sysIntConnect()* to connect its interrupt handler to the interrupt vector.

int_level

This parameter is passed to an external support routine, *sysLanIntEnable()*, which is described below in "External Support Requirements." This routine is called during as part of driver's initialization. It handles any board-specific operations required to allow the servicing of a NIC interrupt on targets that use additional interrupt controller devices to help organize and service the various interrupt sources. This parameter makes it possible for this driver to avoid all board-specific knowledge of such devices.

device restart/reset delay

The global variable *nicRestartDelay* (UINT32), defined in this file, should be initialized in the BSP *sysHwInit()* routine. *nicRestartDelay* is used only with PowerPC platform and is equal to the number of time base increments which makes for 1.6 msec. This corresponds to the delay necessary to respect when restarting or resetting the device.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires several external support functions, defined as macros:

```
SYS_INT_CONNECT(pDrvCtrl, routine, arg)
SYS_INT_DISCONNECT (pDrvCtrl, routine, arg)
SYS_INT_ENABLE(pDrvCtrl)
```

There are default values in the source code for these macros. They presume memory-mapped accesses to the device registers and the normal *intConnect()*, and *intEnable()* BSP functions. The first argument to each is the device controller structure. Thus, each has access back to all the device-specific information. Having the pointer in the macro facilitates the addition of new features to this driver.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector

SEE ALSO **muxLib**

ns16550Sio

NAME **ns16550Sio** – NS 16550 UART tty driver

ROUTINES *ns16550DevInit()* – initialize an NS16550 channel
ns16550IntWr() – handle a transmitter interrupt
ns16550IntRd() – handle a receiver interrupt
ns16550IntEx() – miscellaneous interrupt processing
ns16550Int() – interrupt level processing

DESCRIPTION This is the driver for the NS16552 DUART. This device includes two universal asynchronous receiver/transmitters, a baud rate generator, and a complete modem control capability.

A **NS16550_CHAN** structure is used to describe the serial channel. This data structure is defined in **ns16550Sio.h**.

Only asynchronous serial operation is supported by this driver. The default serial settings are 8 data bits, 1 stop bit, no parity, 9600 baud, and software flow control.

USAGE The BSP's *sysHwInit()* routine typically calls *sysSerialHwInit()*, which creates the **NS16550_CHAN** structure and initializes all the values in the structure (except the **SIO_DRV_FUNCS**) before calling *ns16550DevInit()*. The BSP's *sysHwInit2()* routine typically calls *sysSerialHwInit2()*, which connects the chips interrupts via *intConnect()* (either the single interrupt **ns16550Int** or the three interrupts **ns16550IntWr**, **ns16550IntRd**, and **ns16550IntEx**).

This driver handles setting of hardware options such as parity(odd, even) and number of data bits(5, 6, 7, 8). Hardware flow control is provided with the handshakes RTS/CTS. The function HUPCL(hang up on last close) is available. When hardware flow control is enabled, the signals RTS and DTR are set TRUE and remain set until a HUPCL is performed.

INCLUDE FILES **drv/sio/ns16552Sio.h**

ntEnd

| | |
|--------------------|--|
| NAME | ntEnd – END network interface driver to ULIP for vxSim for Windows NT |
| ROUTINES | <i>ntLoad()</i> – initialize the driver and device <i>ntParse()</i> – parse the init string <i>ntMemInit()</i> – initialize memory for the chip <i>ntPollStart()</i> – start polled mode operations <i>ntPollStop()</i> – stop polled mode operations <i>ntInt()</i> – handle controller interrupt |
| DESCRIPTION | <p>This driver provides a fake ethernet interface to the "ULIP" driver written by WRS. The driver essentially gets packets from vxWorks, and writes them directly to file, where the ULIP driver handles them.</p> <p>The macro <code>SYS_ENET_ADDR_GET</code> is used to get the ethernet address (MAC) for the device. The single argument to this routine is the <code>NTEND_DEVICE</code> pointer. By default this routine copies the ethernet address stored in the global variable <code>ntEnetAddr</code> into the <code>NTEND_DEVICE</code> structure.</p> |
| INCLUDES | end.h endLib.h etherMultiLib.h |
| SEE ALSO | muxLib , <i>endLibWriting and Enhanced Network Driver</i> |

ntPassFsLib

| | |
|-------------------------------|---|
| NAME | ntPassFsLib – pass-through (to Windows NT) file system library |
| ROUTINES | <i>ntPassFsDevInit()</i> – associate a device with ntPassFs file system functions <i>ntPassFsInit()</i> – prepare to use the ntPassFs library |
| DESCRIPTION | <p>This module is only used with VxSim simulated versions of VxWorks.</p> <p>This library provides services for file-oriented device drivers to use the Windows NT file standard. In general, the routines in this library are not to be called directly by users, but rather by the VxWorks I/O System.</p> |
| INITIALIZING PASSFSLIB | <p>Before any other routines in ntPassFsLib can be used, the routine <i>ntPassFsInit()</i> must be called to initialize this library. The <i>ntPassFsDevInit()</i> routine associates a device name with the ntPassFsLib functions. The parameter expected by <i>ntPassFsDevInit()</i> is a</p> |

pointer to a name string, to be used to identify the volume/device. This will be part of the pathname for I/O operations which operate on the device. This name will appear in the I/O system device table, which may be displayed using the *iosDevShow()* routine.

As an example:

```
ntPassFsInit (1);  
ntPassFsDevInit ("host:");
```

After the *ntPassFsDevInit()* call has been made, when **ntPassFsLib** receives a request from the I/O system, it calls the Windows NT I/O system to service the request. Only one volume may be created.

READING DIRECTORY ENTRIES

Directories on a ntPassFs volume may be searched using the *opendir()*, *readdir()*, *rewinddir()*, and *closedir()* routines. These calls allow the names of files and sub-directories to be determined.

To obtain more detailed information about a specific file, use the *fstat()* or *stat()* function. Along with standard file information, the structure used by these routines also returns the file attribute byte from a ntPassFs directory entry.

FILE DATE AND TIME

Windows NT file date and time are passed through to VxWorks.

INCLUDE FILES *ntPassFsLib.h*

SEE ALSO *ioLib*, *iosLib*, *dirLib*, *ramDrv*

ospfLib

NAME *ospfLib* – OSPF version 2 (RFC 1583) routing facilities (OSPF Opt.)

ROUTINES *m2OspfGeneralGroupGet()* – get values of OSPF general group objects (OSPF Opt.)
m2OspfGeneralGroupSet() – set values of OSPF general group objects (OSPF Opt.)
m2OspfAreaEntryGet() – get an entry from the OSPF area table (OSPF Opt.)
m2OspfAreaEntrySet() – set values in an OSPF area entry (OSPF Opt.)
m2OspfStubAreaEntryGet() – get an OSPF stub area entry (OSPF Opt.)
m2OspfStubAreaEntrySet() – set values in an OSPF stub area entry (OSPF Opt.)
m2OspfLsdbEntryGet() – get an OSPF link state database entry (OSPF Opt.)
m2OspfAreaRangeEntryGet() – get an OSPF area range entry (OSPF Opt.)
m2OspfAreaRangeEntrySet() – set values in an OSPF area range entry (OSPF Opt.)
m2OspfHostEntryGet() – get an OSPF host entry (OSPF Opt.)
m2OspfHostEntrySet() – set values in an OSPF host entry (OSPF Opt.)

ospfLib

m2OspfIfEntryGet() – get an OSPF interface entry (OSPF Opt.)
m2OspfIfEntrySet() – set values in an OSPF interface entry (OSPF Opt.)
m2OspfIfMetricEntryGet() – get an OSPF interface metric entry (OSPF Opt.)
m2OspfIfMetricEntrySet() – set OSPF interface metric entry values (OSPF Opt.)
m2OspfVirtIfEntryGet() – get an OSPF virtual interface entry (OSPF Opt.)
m2OspfVirtIfEntrySet() – set OSPF virtual interface entry values (OSPF Opt.)
m2OspfNbrEntryGet() – get an OSPF neighbor entry (OSPF Opt.)
m2OspfNbrEntrySet() – set values in an OSPF neighbor entry (OSPF Opt.)
m2OspfVirtNbrEntryGet() – get an OSPF virtual neighbor entry (OSPF Opt.)
ospfExtRouteAdd() – import external route into OSPF domain (OSPF Opt.)
ospfExtRouteDelete() – delete external route imported into OSPF (OSPF Opt.)
ospfInit() – function to initialize OSPF routing (OSPF Opt.)
ospfNbmaDstAdd() – add NBMA destination
ospfNbmaDstDelete() – delete NBMA destination
ospfTerminate() – free OSPF resources and delete OSPF tasks

DESCRIPTION

This module implements OSPF Version 2 as specified in (RFC 1583). In addition to implementing the routing tasks, this module includes RFC 1253 compliant interfaces that you can use to configure the OSPF MIBs. These may be invoked directly or called by the relevant method routines of an SNMP agent.

To include OSPF in your image you must first define the `INCLUDE_OSPF` in `configAll.h`. Once the system is up and running you need to invoke the `ospfInit()` call. This call has the following structure:

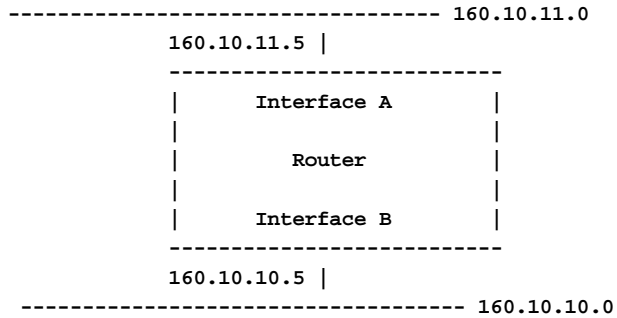
```

STATUS ospfInit
(
    int priority,          /* priority of tasks */
    int options,          /* ospf task options */
    int stackSize,       /* task stack size */
    int routerId         /* the ID for this router */
    FUNCPTR ospfAuthHook /* authentication hook */
)
  
```

After OSPF is up and running, you should configure the OSPF MIB by using the various `m2Ospf` routines. The parameters to these routines are specified in the OSPF MIB as defined in RFC 1253. Explanations for each of the variables may be obtained from the RFC. For additional information on the MIB-II interfaces, please see the manual pages.

EXAMPLE

This section presents a sample configuration as well as the code necessary to make the example work. In the example system, a router is attached to two subnets 160.10.10.0 and 160.10.11.0 with 0xfffff00 as the subnet mask. The interface addresses are 160.10.10.5 and 160.10.11.5.



To set this up programmatically, you would execute the following code:

```

void ospfSetup ()
{
    /* This is a generic setup for all interfaces in the system. */
    M2_OSPF_AREA_ENTRY area;
    M2_OSPF_IF_ENTRY intf;
    area.ospfAreaId = 0x2; /* using area id 2 */
    area.ospfAuthType = 0; /* no authentication */
    if (m2OspfAreaEntrySet (M2_OSPF_AREA_ID |
        M2_OSPF_AUTH_TYPE, &area) != OK)
    {
        return (ERROR);
    };
    /* First we set up Interface A */
    /* set the interface address */
    intf.ospfIfIpAddress = 0xa00a0a05; /* 160.10.10.5 */

    /* address less interface is false */
    intf.ospfAddressLessIf = 0;

    /* interface area id set to 2 */
    intf.ospfIfAreaId = 2;

    /* router priority */
    intf.ospfIfRtrPriority = 5;

    /* various time intervals */
    intf.ospfIfTransitDelay = 1;
    intf.ospfIfRetransInterval = 3;
    intf.ospfIfHelloInterval = 10;
    intf.ospfIfRtrDeadInterval = 40;
    intf.ospfIfPollInterval = 30;

    /* enable OSPF on interface */
    intf.ospfIfAdminStat = M2_ospfAdminStat_enabled;
}

```

```

/* set the parameters for this interface */
if(m2OspfIfEntrySet (M2_OSPF_IF_AREA_ID |
    M2_OSPF_IF_RTR_PRIORITY |
    M2_OSPF_IF_RETRANS_INTERVAL |
    M2_OSPF_IF_HELLO_INTERVAL |
    M2_OSPF_IF_RTR_DEAD_INTERVAL |
    M2_OSPF_IF_POLL_INTERVAL |
    M2_OSPF_IF_ADMIN_STAT,
    &intf) != OK)
{
    return (ERROR);
}

/* similar sequence for Interface B */
intf.ospfIfIpAddress = 0xa00a0b05; /* 160.10.11.5 */
intf.ospfAddressLessIf = 0;
intf.ospfIfAreaId = 2;
intf.ospfIfRtrPriority = 0;
intf.ospfIfTransitDelay = 1;
intf.ospfIfRetransInterval = 3;
intf.ospfIfHelloInterval = 10;
intf.ospfIfRtrDeadInterval = 40;
intf.ospfIfPollInterval = 30;
intf.ospfIfAdminStat = 1;

if (m2OspfIfEntrySet (M2_OSPF_IF_AREA_ID |
    M2_OSPF_IF_RTR_PRIORITY |
    M2_OSPF_IF_RETRANS_INTERVAL |
    M2_OSPF_IF_HELLO_INTERVAL |
    M2_OSPF_IF_RTR_DEAD_INTERVAL |
    M2_OSPF_IF_POLL_INTERVAL |
    M2_OSPF_IF_ADMIN_STAT, &intf) != OK)
{
    return (ERROR);
}

```

After this code has executed, the system is set up to use OSPF to route between the two interfaces (A and B). The system will now continue to participate in the OSPF routing protocol until either the system is shut off or further calls are made into the system using the m2[*] interfaces. Note that it may not be necessary to set all the parameters as shown above if the default value of the parameter is acceptable for your configuration. Default values are as specified in the MIB (RFC 1253).

INCLUDE FILES **ospfLib.h**

SEE ALSO RFC 1583 and RFC 1253

passFsLib

| | |
|--------------------|---|
| NAME | passFsLib – pass-through (to UNIX) file system library (VxSim) |
| ROUTINES | <i>passFsDevInit()</i> – associate a device with passFs file system functions <i>passFsInit()</i> – prepare to use the passFs library |
| DESCRIPTION | This module is only used with VxSim simulated versions of VxWorks. This library provides services for file-oriented device drivers to use the UNIX file standard. This module takes care of all the buffering, directory maintenance, and file system details that are necessary. In general, the routines in this library are not to be called directly by users, but rather by the VxWorks I/O System. |

INITIALIZING PASSFSLIB

Before any other routines in **passFsLib** can be used, the routine *passFsInit()* must be called to initialize this library. The *passFsDevInit()* routine associates a device name with the **passFsLib** functions. The parameter expected by *passFsDevInit()* is a pointer to a name string, to be used to identify the volume/device. This will be part of the pathname for I/O operations which operate on the device. This name will appear in the I/O system device table, which may be displayed using the *iosDevShow()* routine.

As an example:

```
passFsInit (1);  
passFsDevInit ("host:");
```

After the *passFsDevInit()* call has been made, when **passFsLib** receives a request from the I/O system, it calls the UNIX I/O system to service the request. Only one volume may be created.

READING DIRECTORY ENTRIES

Directories on a passFs volume may be searched using the *opendir()*, *readdir()*, *rewinddir()*, and *closedir()* routines. These calls allow the names of files and sub-directories to be determined.

To obtain more detailed information about a specific file, use the *fstat()* or *stat()* function. Along with standard file information, the structure used by these routines also returns the file attribute byte from a passFs directory entry.

FILE DATE AND TIME

UNIX file date and time are passed though to VxWorks.

INCLUDE FILES **passFsLib.h**

SEE ALSO **ioLib, iosLib, dirLib, ramDrv**

pccardLib

| | |
|--------------------|--|
| NAME | pccardLib – PC CARD enabler library |
| ROUTINES | <p><i>pccardMount()</i> – mount a DOS file system</p> <p><i>pccardMkfs()</i> – initialize a device and mount a DOS file system</p> <p><i>pccardAtaEnabler()</i> – enable the PCMCIA-ATA device</p> <p><i>pccardSramEnabler()</i> – enable the PCMCIA-SRAM driver</p> <p><i>pccardEltEnabler()</i> – enable the PCMCIA Etherlink III card</p> <p><i>pccardTffsEnabler()</i> – enable the PCMCIA-TFFS driver</p> |
| DESCRIPTION | <p>This library provides generic facilities for enabling PC CARD. Each PC card device driver needs to provide an enabler routine and a CSC interrupt handler. The enabler routine must be in the pccardEnabler structure. Each PC card driver has its own resource structure, xxResources. The ATA PC card driver resource structure is ataResources in sysLib, which also supports a local IDE disk. The resource structure has a PC card common resource structure in the first member. Other members are device-driver dependent resources.</p> <p>The PCMCIA chip initialization routines <i>tcicInit()</i> and <i>pcicInit()</i> are included in the PCMCIA chip table pcmciaAdapter. This table is scanned when the PCMCIA library is initialized. If the initialization routine finds the PCMCIA chip, it registers all function pointers of the PCMCIA_CHIP structure.</p> <p>A memory window defined in pcmciaMemwin is used to access the CIS of a PC card through the routines in cisLib.</p> |
| SEE ALSO | pcmciaLib , cisLib , tcic , pcic |

pcic

| | |
|--------------------|---|
| NAME | pcic – Intel 82365SL PCMCIA host bus adaptor chip library |
| ROUTINES | <i>pcicInit()</i> – initialize the PCIC chip |
| DESCRIPTION | <p>This library contains routines to manipulate the PCMCIA functions on the Intel 82365 series PCMCIA chip. The following compatible chips are also supported:</p> <ul style="list-style-type: none"> – Cirrus Logic PD6712/20/22 – Vadem VG468 – VLSI 82c146 – Ricoh RF5C series |

The initialization routine *pcicInit()* is the only global function and is included in the PCMCIA chip table **pcmciaAdapter**. If *pcicInit()* finds the PCIC chip, it registers all function pointers of the **PCMCIA_CHIP** structure.

pcicShow

| | |
|--------------------|---|
| NAME | pcicShow – Intel 82365SL PCMCIA host bus adaptor chip show library |
| ROUTINES | <i>pcicShow()</i> – show all configurations of the PCIC chip |
| DESCRIPTION | This is a driver show routine for the Intel 82365 series PCMCIA chip. <i>pcicShow()</i> is the only global function and is installed in the PCMCIA chip table pcmciaAdapter in <i>pcmciaShowInit()</i> . |
| SEE ALSO | pcicShow |

pcmciaLib

| | |
|--------------------|---|
| NAME | pcmciaLib – generic PCMCIA event-handling facilities |
| ROUTINES | <i>pcmciaInit()</i> – initialize the PCMCIA event-handling package <i>pcmciaad()</i> – handle task-level PCMCIA events |
| DESCRIPTION | This library provides generic facilities for handling PCMCIA events. |

USER-CALLABLE ROUTINES

Before the driver can be used, it must be initialized by calling *pcmciaInit()*. This routine should be called exactly once, before any PC card device driver is used. Normally, it is called from *usrRoot()* in **usrConfig.c**.

The *pcmciaInit()* routine performs the following actions:

- Creates a message queue.
- Spawns a PCMCIA daemon, which handles jobs in the message queue.
- Finds out which PCMCIA chip is installed and fills out the **PCMCIA_CHIP** structure.
- Connects the CSC (Card Status Change) interrupt handler.
- Searches all sockets for a PC card. If a card is found, it:

- gets CIS (Card Information Structure) information from a card
- determines what type of PC card is in the socket
- allocates a resource for the card if the card is supported
- enables the card

- Enables the CSC interrupt.

The CSC interrupt handler performs the following actions:

- Searches all sockets for CSC events.
- Calls the PC card's CSC interrupt handler, if there is a PC card in the socket.
- If the CSC event is a hot insertion, it asks the PCMCIA daemon to call *cisGet()* at task level. This call reads the CIS, determines the type of PC card, and initializes a device driver for the card.
- If the CSC event is a hot removal, it asks the PCMCIA daemon to call *cisFree()* at task level. This call de-allocates resources.

pcmciaShow

| | |
|--------------------|---|
| NAME | pcmciaShow – PCMCIA show library |
| ROUTINES | <i>pcmciaShowInit()</i> – initialize all show routines for PCMCIA drivers <i>pcmciaShow()</i> – show all configurations of the PCMCIA chip |
| DESCRIPTION | This library provides a show routine that shows the status of the PCMCIA chip and the PC card. |

pentiumALib

| | |
|-----------------|--|
| NAME | pentiumALib – Pentium and PentiumPro specific routines |
| ROUTINES | <i>pentiumCr4Get()</i> – Get a content of CR4 register <i>pentiumCr4Set()</i> – Set a specified value to CR4 register <i>pentiumPmcStart()</i> – start both PMC0 and PMC1 <i>pentiumPmcStop()</i> – stop both PMC0 and PMC1 <i>pentiumPmcStop1()</i> – stop PMC1 <i>pentiumPmcGet()</i> – get contents of PMC0 and PMC1 <i>pentiumPmcGet0()</i> – get a content of PMC0 <i>pentiumPmcGet1()</i> – get a content of PMC1 |

pentiumPmcReset() – reset both PMC0 and PMC1
pentiumPmcReset0() – reset PMC0
pentiumPmcReset1() – reset PMC1
pentiumTscGet64() – get 64Bit TSC (Timestamp Counter)
pentiumTscGet32() – get a lower half of the 64Bit TSC (Timestamp Counter)
pentiumTscReset() – reset the TSC (Timestamp Counter)
pentiumMsrGet() – get a content of the specified MSR (Model Specific Register)
pentiumMsrSet() – set a value to the specified MSR (Model Specific Registers)
pentiumTlbFlush() – flush TLBs (Translation Lookaside Buffers)
pentiumSerialize() – execute a serializing instruction CPUID
pentiumBts() – execute atomic compare-and-exchange instruction to set a bit
pentiumBtc() – execute atomic compare-and-exchange instruction to clear a bit

DESCRIPTION This module contains Pentium and PentiumPro specific routines written in assembly language.

MCA (Machine Check Architecture)

The Pentium processor introduced a new exception called the machine-check exception (interrupt-18). This exception is used to signal hardware-related errors, such as a parity error on a read cycle. The PentiumPro processor extends the types of errors that can be detected and that generate a machine-check exception. It also provides a new machine-check architecture that records information about a machine-check error and provides the basis for an extended error logging capability.

MCA is enabled and its status registers are cleared zero in *sysHwInit()*. Its registers are accessed by *pentiumMsrSet()* and *pentiumMsrGet()*.

PMC (Performance Monitoring Counters)

The PentiumPro processor has two performance-monitoring counters for use in monitoring internal hardware operations. These counters are duration or event counters that can be programmed to count any of approximately 100 different types of events, such as the number of instructions decoded, number of interrupts received, or number of cache loads.

There are nine routines to interface the PMC. These nine routines are:

```

STATUS pentiumPmcStart
(
    int pmcEvtSel0;          /* performance event select register 0 */
    int pmcEvtSel1;          /* performance event select register 1 */
)

void pentiumPmcStop (void)

void pentiumPmcStop1 (void)

void pentiumPmcGet
  
```

pentiumALib

```

(
    long long int * pPmc0; /* performance monitoring counter 0 */
    long long int * pPmc1; /* performance monitoring counter 1 */
)

void pentiumPmcGet0
(
    long long int * pPmc0; /* performance monitoring counter 0 */
)

void pentiumPmcGet1
(
    long long int * pPmc1; /* performance monitoring counter 1 */
)

void pentiumPmcReset (void)

void pentiumPmcReset0 (void)

void pentiumPmcReset1 (void)

```

pentiumPmcStart() starts both PMC0 and PMC1. *pentiumPmcStop()* stops them, and *pentiumPmcStop1()* stops only PMC1. *pentiumPmcGet()* gets contents of PMC0 and PMC1. *pentiumPmcGet0()* gets a content of PMC0, and *pentiumPmcGet1()* gets a content of PMC1. *pentiumPmcReset()* resets both PMC0 and PMC1. *pentiumPmcReset0()* resets PMC0, and *pentiumPmcReset1()* resets PMC1. PMC is enabled in *sysHwInit()*. Selected events in the default configuration are PMC0 = number of hardware interrupts received and PMC1 = number of misaligned data memory references.

MSR (Model Specific Register)

The concept of model-specific registers (MSRs) to control hardware functions in the processor or to monitor processor activity was introduced in the PentiumPro processor. The new registers control the debug extensions, the performance counters, the machine-check exception capability, the machine check architecture, and the MTRRs. The MSRs can be read and written to using the RDMSR and WRMSR instructions, respectively.

There are two routines to interface the MSR. These two routines are:

```

void pentiumMsrGet
(
    int address,          /* MSR address */
    long long int * pData /* MSR data */
)

void pentiumMsrSet

```

```
(
  int address,           /* MSR address */
  long long int * pData /* MSR data */
)
```

pentiumMsrGet() get a content of the specified MSR, and *pentiumMsrSet()* set a value to the specified MSR.

TSC (Time Stamp Counter)

The PentiumPro processor provides a 64-bit time-stamp counter that is incremented every processor clock cycle. The counter is incremented even when the processor is halted by the HLT instruction or the external STPCLK# pin. The time-stamp counter is set to 0 following a hardware reset of the processor. The RDTSC instruction reads the time stamp counter and is guaranteed to return a monotonically increasing unique value whenever executed, except for 64-bit counter wraparound. Intel guarantees, architecturally, that the time-stamp counter frequency and configuration will be such that it will not wraparound within 10 years after being reset to 0. The period for counter wrap is several thousands of years in the PentiumPro and Pentium processors.

There are three routines to interface the TSC. These three routines are:

```
void pentiumTscReset (void)

void pentiumTscGet32 (void)

void pentiumTscGet64
(
  long long int * pTsc /* TSC */
)
```

pentiumTscReset() reset the TSC. *pentiumTscGet32()* gets a lower half of the 64Bit TSC, and *pentiumTscGet64()* gets whole 64Bit TSC.

Four other routines are provided in this library. They are:

```
void pentiumTlbFlush (void)

void pentiumSerialize (void)

STATUS pentiumBts
(
  char * pFlag /* flag address */
)

STATUS pentiumBtc (pFlag)
(
  char * pFlag /* flag address */
)
```

pentiumTlbFlush() flushes TLBs (Translation Lookaside Buffers). *pentiumSerialize()* does serialization by executing CPUID instruction. *pentiumBts()* executes an atomic compare-and-exchange instruction to set a bit. *pentiumBtc()* executes an atomic compare-and-exchange instruction to clear a bit.

SEE ALSO *Pentium, PentiumPro Family Developer's Manual*

pentiumLib

NAME *pentiumLib* – Pentium and PentiumPro library

ROUTINES *pentiumMtrrEnable()* – enable MTRR (Memory Type Range Register)
pentiumMtrrDisable() – disable MTRR (Memory Type Range Register)
pentiumMtrrGet() – get MTRRs to a specified MTRR table
pentiumMtrrSet() – set MTRRs from specified MTRR table with WRMSR instruction.

DESCRIPTION This library provides Pentium and PentiumPro specific routines.

MTRR (Memory Type Range Register) are a new feature introduced in the PentiumPro processor that allow the processor to optimize memory operations for different types of memory, such as RAM, ROM, frame buffer memory, and memory-mapped IO. MTRRs configure an internal map of how physical address ranges are mapped to various types of memory. The processor uses this internal map to determine the cacheability of various physical memory locations and the optimal method of accessing memory locations. For example, if a memory location is specified in an MTRR as write-through memory, the processor handles accesses to this location as follows. It reads data from that location in lines and caches the read data or maps all writes to that location to the bus and updates the cache to maintain cache coherency. In mapping the physical address space with MTRRs, the processor recognizes five types of memory: uncacheable (UC), write-combining (WC), write-through (WT), write-protected (WP), and write-back (WB).

There are one table – *sysMtrr[]* in *sysLib.c* – and four routines to interface the MTRR. These four routines are:

```
void pentiumMtrrEnable (void)

void pentiumMtrrDisable (void)

void pentiumMtrrGet
(
    MTRR * pMtrr          /* MTRR table */
)

void pentiumMtrrSet (void)
```

```
(  
    MTRR * pMtrr          /* MTRR table */  
)
```

pentiumMtrrEnable() enables MTRR, *pentiumMtrrDisable()* disables MTRR. *pentiumMtrrGet()* gets MTRRs to the specified MTRR table. *pentiumMtrrSet()* sets MTRRs from the specified MTRR table. The MTRR table is defined as follows:

```
typedef struct mtrr_fix      /* MTRR - fixed range register */  
{  
    char type[8];          /* address range: [0]=0-7 ... [7]=56-63 */  
} MTRR_FIX;  
  
typedef struct mtrr_var     /* MTRR - variable range register */  
{  
    long long int base;    /* base register */  
    long long int mask;    /* mask register */  
} MTRR_VAR;  
  
typedef struct mtrr        /* MTRR */  
{  
    int cap[2];           /* MTRR cap register */  
    int deftype[2];       /* MTRR defType register */  
    MTRR_FIX fix[11];     /* MTRR fixed range registers */  
    MTRR_VAR var[8];      /* MTRR variable range registers */  
} MTRR;
```

Fixed Range Register's type array can be one of following memory types. **MTRR_UC** (uncacheable), **MTRR_WC** (write-combining), **MTRR_WT** (write-through), **MTRR_WP** (write-protected), and **MTRR_WB** (write-back). MTRR is enabled in *sysHwInit()*.

SEE ALSO

Pentium, PentiumPro Family Developer's Manual

pentiumShow

| | |
|--------------------|--|
| NAME | pentiumShow – Pentium and PentiumPro specific show routines |
| ROUTINES | <i>pentiumMcaShow()</i> – show MCA (Machine Check Architecture) registers <i>pentiumPmcShow()</i> – show PMCs (Performance Monitoring Counters) |
| DESCRIPTION | This library provides Pentium and PentiumPro specific show routines. <i>pentiumMcaShow()</i> shows Machine Check Global Control Registers and Error Reporting Register Banks. <i>pentiumPmcShow()</i> shows PMC0 and PMC1, and reset them if the parameter zap is TRUE. |
| SEE ALSO | <i>VxWorks Programmer's Guide: Configuration</i> |

pingLib

| | |
|--------------------|---|
| NAME | pingLib – Packet InterNet Grouper (PING) library |
| ROUTINES | <i>pingLibInit()</i> – initialize the <i>ping()</i> utility <i>ping()</i> – test that a remote host is reachable |
| DESCRIPTION | This library contains the <i>ping()</i> utility, which tests the reachability of a remote host. The routine <i>ping()</i> is typically called from the VxWorks shell to check the network connection to another VxWorks target or to a UNIX host. <i>ping()</i> may also be used programmatically by applications that require such a test. The remote host must be running TCP/IP networking code that responds to ICMP echo request packets. The <i>ping()</i> routine is re-entrant, thus may be called by many tasks concurrently. The routine <i>pingLibInit()</i> initializes the <i>ping()</i> utility and allocates resources used by this library. It is called automatically when the configuration macro <code>INCLUDE_PING</code> is defined. |

pipeDrv

NAME `pipeDrv` – pipe I/O driver

ROUTINES `pipeDrv()` – initialize the pipe driver
`pipeDevCreate()` – create a pipe device

DESCRIPTION The pipe driver provides a mechanism that lets tasks communicate with each other through the standard I/O interface. Pipes can be read and written with normal `read()` and `write()` calls. The pipe driver is initialized with `pipeDrv()`. Pipe devices are created with `pipeDevCreate()`.

The pipe driver uses the VxWorks message queue facility to do the actual buffering and delivering of messages. The pipe driver simply provides access to the message queue facility through the I/O system. The main differences between using pipes and using message queues directly are:

- pipes are named (with I/O device names).
- pipes use the standard I/O functions -- `open()`, `close()`, `read()`, `write()` -- while message queues use the functions `msgQSend()` and `msgQReceive()`.
- pipes respond to standard `ioctl()` functions.
- pipes can be used in a `select()` call.
- message queues have more flexible options for timeouts and message priorities.
- pipes are less efficient than message queues because of the additional overhead of the I/O system.

INSTALLING THE DRIVER

Before using the driver, it must be initialized and installed by calling `pipeDrv()`. This routine must be called before any pipes are created. It is called automatically by the root task, `usrRoot()`, in `usrConfig.c` when the configuration macro `INCLUDE_PIPES` is defined.

CREATING PIPES Before a pipe can be used, it must be created with `pipeDevCreate()`. For example, to create a device pipe `"/pipe/demo"` with up to 10 messages of size 100 bytes, the proper call is:

```
pipeDevCreate ("/pipe/demo", 10, 100);
```

USING PIPES Once a pipe has been created it can be opened, closed, read, and written just like any other I/O device. Often the data that is read and written to a pipe is a structure of some type. Thus, the following example writes to a pipe and reads back the same data:

```
{  
    int fd;  
    struct msg outMsg;
```

```
struct msg inMsg;
int len;
fd = open ("/pipe/demo", O_RDWR);
write (fd, &outMsg, sizeof (struct msg));
len = read (fd, &inMsg, sizeof (struct msg));
close (fd);
}
```

The data written to a pipe is kept as a single message and will be read all at once in a single read. If *read()* is called with a buffer that is smaller than the message being read, the remainder of the message will be discarded. Thus, pipe I/O is "message oriented" rather than "stream oriented." In this respect, VxWorks pipes differ significantly from UNIX pipes which are stream oriented and do not preserve message boundaries.

WRITING TO PIPES FROM INTERRUPT SERVICE ROUTINES

Interrupt service routines (ISR) can write to pipes, providing one of several ways in which ISRs can communicate with tasks. For example, an interrupt service routine may handle the time-critical interrupt response and then send a message on a pipe to a task that will continue with the less critical aspects. However, the use of pipes to communicate from an ISR to a task is now discouraged in favor of the direct message queue facility, which offers lower overhead (see the manual entry for *msgQLib* for more information).

SELECT CALLS

An important feature of pipes is their ability to be used in a *select()* call. The *select()* routine allows a task to wait for input from any of a selected set of I/O devices. A task can use *select()* to wait for input from any combination of pipes, sockets, or serial devices. See the manual entry for *select()*.

IOCTL FUNCTIONS

Pipe devices respond to the following *ioctl()* functions. These functions are defined in the header file *ioLib.h*.

FIOGETNAME

Gets the file name of *fd* and copies it to the buffer referenced by *nameBuf*:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

FIONREAD

Copies to *nBytesUnread* the number of bytes remaining in the first message in the pipe:

```
status = ioctl (fd, FIONREAD, &nBytesUnread);
```

FIONMSGS

Copies to *nMessages* the number of discrete messages remaining in the pipe:

```
status = ioctl (fd, FIONMSGS, &nMessages);
```

FIOFLUSH

Discards all messages in the pipe and releases the memory block that contained them:

```
status = ioctl (fd, FIOFLUSH, 0);
```

INCLUDE FILES **ioLib.h, pipeDrv.h**

SEE ALSO *select()*, **msgQLib**, *VxWorks Programmer's Guide: I/O System*

ppc403Sio

NAME **ppc403Sio** – ppc403GA serial driver

ROUTINES *ppc403DummyCallback()* – dummy callback routine
ppc403DevInit() – initialize the serial port unit
ppc403IntWr() – handle a transmitter interrupt
ppc403IntRd() – handle a receiver interrupt
ppc403IntEx() – handle error interrupts

DESCRIPTION This is the driver for PPC403GA serial port on the on-chip peripheral bus. The SPU (serial port unit) consists of three main elements: receiver, transmitter, and baud-rate generator. For details, refer to the *PPC403GA Embedded Controller User's Manual*.

USAGE A **PPC403_CHAN** structure is used to describe the chip. This data structure contains the single serial channel. The BSP's *sysHwInit()* routine typically calls *sysSerialHwInit()* which initializes all the values in the **PPC403_CHAN** structure (except the **SIO_DRV_FUNCS**) before calling *ppc403DevInit()*. The BSP's *sysHwInit2()* routine typically calls *sysSerialHwInit2()* which connects the chip interrupt routines *ppc403IntWr()* and *ppc403IntRd()* via *intConnect()*.

IOCTL FUNCTIONS This driver responds to the same *ioctl()* codes as other SIO drivers; for more information, see **sioLib.h**.

INCLUDE FILES **drv/sio/ppc403Sio.h**

ppc860Sio

| | |
|----------------------|--|
| NAME | ppc860Sio – Motorola MPC800 SMC UART serial driver |
| ROUTINES | <i>ppc860DevInit()</i> – initialize the SMC <i>ppc860Int()</i> – handle an SMC interrupt |
| DESCRIPTION | This is the driver for the SMCs in the internal Communications Processor (CP) of the Motorola MPC68860/68821. This driver only supports the SMCs in asynchronous UART mode. |
| USAGE | A <code>PPC800SMC_CHAN</code> structure is used to describe the chip. The BSP's <i>sysHwInit()</i> routine typically calls <i>sysSerialHwInit()</i> , which initializes all the values in the <code>PPC860SMC_CHAN</code> structure (except the <code>SIO_DRV_FUNCS</code>) before calling <i>ppc860DevInit()</i> . The BSP's <i>sysHwInit2()</i> routine typically calls <i>sysSerialHwInit2()</i> which connects the chip's interrupts via <i>intConnect()</i> . |
| INCLUDE FILES | <code>drv/sio/ppc860Sio.h</code> |

pppHookLib

| | |
|----------------------|---|
| NAME | pppHookLib – PPP hook library |
| ROUTINES | <i>pppHookAdd()</i> – add a hook routine on a unit basis <i>pppHookDelete()</i> – delete a hook routine on a unit basis |
| DESCRIPTION | This library provides routines to add and delete connect and disconnect routines. The connect routine, added on a unit basis, is called before the initial phase of link option negotiation. The disconnect routine, added on a unit basis is called before the PPP connection is closed. These connect and disconnect routines can be used to hook up additional software. If either connect or disconnect hook returns <code>ERROR</code> , the connection is terminated immediately. This library is automatically linked into the VxWorks system image when the configuration macro <code>INCLUDE_PPP</code> is defined. |
| INCLUDE FILES | <code>pppLib.h</code> |
| SEE ALSO | pppLib , <i>VxWorks Programmer's Guide: Network</i> |

pppLib

NAME **pppLib** – Point-to-Point Protocol library

ROUTINES *pppInit()* – initialize a PPP network interface
pppDelete() – delete a PPP network interface

DESCRIPTION This library implements the VxWorks Point-to-Point Protocol (PPP) facility. PPP allows VxWorks to communicate with other machines by sending encapsulated multi-protocol datagrams over a point-to-point serial link. VxWorks may have up to 16 PPP interfaces active at any one time. Each individual interface (or "unit") operates independent of the state of other PPP units.

USER-CALLABLE ROUTINES

PPP network interfaces are initialized using the *pppInit()* routine. This routine's parameters specify the unit number, the name of the serial interface (*tty*) device, Internet (IP) addresses for both ends of the link, the interface baud rate, an optional pointer to a configuration options structure, and an optional pointer to a configuration options file. The *pppDelete()* routine deletes a specified PPP interface.

DATA ENCAPSULATION

PPP uses HDLC-like framing, in which five header and three trailer octets are used to encapsulate each datagram. In environments where bandwidth is at a premium, the total encapsulation may be shortened to four octets with the available address/control and protocol field compression options.

LINK CONTROL PROTOCOL

PPP incorporates a link-layer protocol called Link Control Protocol (LCP), which is responsible for the link set up, configuration, and termination. LCP provides for automatic negotiation of several link options, including datagram encapsulation format, user authentication, and link monitoring (LCP echo request/reply).

NETWORK CONTROL PROTOCOLS

PPP's Network Control Protocols (NCP) allow PPP to support different network protocols. VxWorks supports only one NCP, the Internet Protocol Control Protocol (IPCP), which allows the establishment and configuration of IP over PPP links. IPCP supports the negotiation of IP addresses and TCP/IP header compression (commonly called "VJ" compression).

AUTHENTICATION The VxWorks PPP implementation supports two separate user authentication protocols: the Password Authentication Protocol (PAP) and the Challenge-Handshake Authentication Protocol (CHAP). While PAP only authenticates at the time of link establishment, CHAP may be configured to periodically require authentication

throughout the life of the link. Both protocols are independent of one another, and either may be configured in through the PPP options structure or options file.

IMPLEMENTATION Each VxWorks PPP interface is handled by two tasks: the daemon task (*tPPPunit*) and the write task (*tPPPunitWrt*).

The daemon task controls the various PPP control protocols (LCP, IPCP, CHAP, and PAP). Each PPP interface has its own daemon task that handles link set up, negotiation of link options, link-layer user authentication, and link termination. The daemon task is not used for the actual sending and receiving of IP datagrams.

The write task controls the transmit end of a PPP driver interface. Each PPP interface has its own write task that handles the actual sending of a packet by writing data to the *tty* device. Whenever a packet is ready to be sent out, the PPP driver activates this task by giving a semaphore. The write task then completes the packet framing and writes the packet data to the *tty* device.

The receive end of the PPP interface is implemented as a "hook" into the *tty* device driver. The *tty* driver's receive interrupt service routine (ISR) calls the PPP driver's ISR every time a character is received on the serial channel. When the correct PPP framing character sequence is received, the PPP ISR schedules the *tNetTask* task to call the PPP input routine. The PPP input routine reads a whole PPP packet out of the *tty* ring buffer and processes it according to PPP framing rules. The packet is then queued either to the IP input queue or to the PPP daemon task input queue.

INCLUDE FILES **pppLib.h**

SEE ALSO **ifLib**, **tyLib**, **pppSecretLib**, **pppShow**, *VxWorks Programmer's Guide: Network*, *RFC-1332: The PPP Internet Protocol Control Protocol (IPCP)*, *RFC-1334: PPP Authentication Protocols*, *RFC-1548: The Point-to-Point Protocol (PPP)*, *RFC-1549: PPP in HDLC Framing*

ACKNOWLEDGEMENT

This program is based on original work done by Paul Mackerras of Australian National University, Brad Parker, Greg Christy, Drew D. Perkins, Rick Adams, and Chris Torek.

pppSecretLib

| | |
|----------------------|---|
| NAME | pppSecretLib – PPP authentication secrets library |
| ROUTINES | <i>pppSecretAdd()</i> – add a secret to the PPP authentication secrets table <i>pppSecretDelete()</i> – delete a secret from the PPP authentication secrets table |
| DESCRIPTION | <p>This library provides routines to create and manipulate a table of "secrets" for use with Point-to-Point Protocol (PPP) user authentication protocols. The secrets in the secrets table can be searched by peers on a PPP link so that one peer (client) can send a secret word to the other peer (server). If the client cannot find a suitable secret when required to do so, or the secret received by the server is not valid, the PPP link may be terminated.</p> <p>This library is automatically linked into the VxWorks system image when the configuration macro INCLUDE_PPP is defined.</p> |
| INCLUDE FILES | pppLib.h |
| SEE ALSO | pppLib , pppShow , <i>VxWorks Programmer's Guide: Network</i> |

pppShow

| | |
|----------------------|--|
| NAME | pppShow – Point-to-Point Protocol show routines |
| ROUTINES | <i>pppInfoShow()</i> – display PPP link status information <i>pppInfoGet()</i> – get PPP link status information <i>pppstatShow()</i> – display PPP link statistics <i>pppstatGet()</i> – get PPP link statistics <i>pppSecretShow()</i> – display the PPP authentication secrets table |
| DESCRIPTION | <p>This library provides routines to show Point-to-Point Protocol (PPP) link status information and statistics. Also provided are routines that programmatically access this same information.</p> <p>This library is automatically linked into the VxWorks system image when the configuration macro INCLUDE_PPP is defined.</p> |
| INCLUDE FILES | pppLib.h |
| SEE ALSO | pppLib , <i>VxWorks Programmer's Guide: Network</i> |

proxyArpLib

| | |
|----------------------|--|
| NAME | proxyArpLib – proxy Address Resolution Protocol (ARP) library |
| ROUTINES | <i>proxyArpLibInit()</i> – initialize proxy ARP <i>proxyNetCreate()</i> – create a proxy ARP network <i>proxyNetDelete()</i> – delete a proxy network <i>proxyNetShow()</i> – show proxy ARP networks <i>proxyPortFwdOn()</i> – enable broadcast forwarding for a particular port <i>proxyPortFwdOff()</i> – disable broadcast forwarding for a particular port <i>proxyPortShow()</i> – show enabled ports |
| DESCRIPTION | <p>This library provides transparent network access by using the Address Resolution Protocol (ARP) to make logically distinct networks appear as one logical network (that is, the networks share the same address space). This module implements a proxy ARP scheme which provides an alternate method (to subnets) of access to the WRS backplane.</p> <p>This module implements the proxy server. The proxy server is the multi-homed target which provides network transparency over the backplane by watching for and answering ARP requests.</p> <p>This implementation supports only a single tier of backplane networks (that is, only targets on directly attached interfaces are proxied for). Only one proxy server resides on a particular backplane network.</p> <p>This library is initialized by calling <i>proxyArpLibInit()</i>. Proxy networks are created by calling <i>proxyNetCreate()</i> and deleted by calling <i>proxyNetDelete()</i>. The <i>proxyNetShow()</i> routine displays the proxy and main networks and the clients that reside on them.</p> <p>A VxWorks backplane target registers itself as a target (proxy client) on the proxy network by calling <i>proxyReg()</i>. It unregisters itself by calling <i>proxyUnreg()</i>. These routines are provided in proxyLib.</p> <p>To minimize and control backplane (proxy network) broadcast traffic, the proxy server must be configured to pass through broadcasts to a certain set of destination ports. Ports are enabled with the call <i>proxyPortFwdOn()</i> and are disabled with the call <i>proxyPortFwdOff()</i>. To see the ports currently enabled use <i>proxyPortShow()</i>. By default, only the BOOTP server port is enabled.</p> <p>For more information on proxy ARP, see the <i>VxWorks Programmer's Guide: Network</i></p> |
| INCLUDE FILES | proxyArpLib.h |
| SEE ALSO | proxyLib , RFC 925, RFC 1027, RFC 826, <i>Network Programmer's Guide VxWorks Programmer's Guide: Network</i> |

proxyLib

| | |
|----------------------|--|
| NAME | proxyLib – proxy Address Resolution Protocol (ARP) client library |
| ROUTINES | <i>proxyReg()</i> – register a proxy client <i>proxyUnreg()</i> – unregister a proxy client |
| DESCRIPTION | <p>This library implements the client side of the proxy Address Resolution Protocol (ARP). It allows a VxWorks target to register itself as a proxy client by calling <i>proxyReg()</i> and to unregister itself by calling <i>proxyUnreg()</i>.</p> <p>Both commands take an interface name and an IP address as arguments. The interface, <i>ifName</i>, specifies the interface through which to send the message. <i>ifName</i> must be a backplane interface. <i>proxyAddr</i> is the IP address associated with the interface <i>ifName</i>.</p> |
| INCLUDE FILES | proxyArpLib.h |
| SEE ALSO | proxyArpLib , <i>VxWorks Programmer's Guide: Network</i> |

ptyDrv

| | |
|--------------------------------|--|
| NAME | ptyDrv – pseudo-terminal driver |
| ROUTINES | <i>ptyDrv()</i> – initialize the pseudo-terminal driver <i>ptyDevCreate()</i> – create a pseudo terminal |
| DESCRIPTION | <p>The pseudo-terminal driver provides a tty-like interface between a master and slave process, typically in network applications. The master process simulates the "hardware" side of the driver (e.g., a USART serial chip), while the slave process is the application program that normally talks to the driver.</p> |
| USER-CALLABLE ROUTINES | <p>Most of the routines in this driver are accessible only through the I/O system. However, the following routines must be called directly: <i>ptyDrv()</i> to initialize the driver, and <i>ptyDevCreate()</i> to create devices.</p> |
| INITIALIZING THE DRIVER | <p>Before using the driver, it must be initialized by calling <i>ptyDrv()</i>. This routine must be called before any reads, writes, or calls to <i>ptyDevCreate()</i>.</p> |

CREATING PSEUDO-TERMINAL DEVICES

Before a pseudo-terminal can be used, it must be created by calling *ptyDevCreate()*:

```

STATUS ptyDevCreate
(
    char *name,      /* name of pseudo terminal */
    int  rdBufSize, /* size of terminal read buffer */
    int  wrtBufSize /* size of write buffer */
)

```

For instance, to create the device pair `"/pty/0.M"` and `"/pty/0.S"`, with read and write buffer sizes of 512 bytes, the proper call would be:

```
ptyDevCreate ("/pty/0.", 512, 512);
```

When *ptyDevCreate()* is called, two devices are created, a master and slave. One is called *nameM* and the other *nameS*. They can then be opened by the master and slave processes. Data written to the master device can then be read on the slave device, and vice versa. Calls to *ioctl()* may be made to either device, but they should only apply to the slave side, since the master and slave are the same device.

- IOCTL FUNCTIONS** Pseudo-terminal drivers respond to the same *ioctl()* functions used by tty devices. These functions are defined in **ioLib.h** and documented in the manual entry for **tyLib**.
- CAVEAT** Pseudo-terminal devices cannot be deleted and the associated memory cannot be reclaimed.
- INCLUDE FILES** **ioLib.h**, **ptyDrv.h**
- SEE ALSO** **tyLib**, *VxWorks Programmer's Guide: I/O System*

ramDrv

- NAME** **ramDrv** – RAM disk driver
- ROUTINES** *ramDrv()* – prepare a RAM disk driver for use (optional)
ramDevCreate() – create a RAM disk device
- DESCRIPTION** This driver emulates a disk driver, but actually keeps all data in memory. The memory location and size are specified when the "disk" is created. The RAM disk feature is useful when data must be preserved between boots of VxWorks or when sharing data between CPUs.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. Two routines, however, can be called directly by the user. The first, *ramDrv()*, provides no real function except to parallel the initialization function found in true disk device drivers. A call to *ramDrv()* is not required to use the RAM disk driver. However, the second routine, *ramDevCreate()*, must be called directly to create RAM disk devices.

Once the device has been created, it must be associated with a name and file system (dosFs, rt11Fs, or rawFs). This is accomplished by passing the value returned by *ramDevCreate()*, a pointer to a block device structure, to the file system's device initialization routine or make-file-system routine. See the manual entry *ramDevCreate()* for a more detailed discussion.

IOCTL FUNCTIONS The RAM driver is called in response to *ioctl()* codes in the same manner as a normal disk driver. When the file system is unable to handle a specific *ioctl()* request, it is passed to the **ramDrv** driver. Although there is no physical device to be controlled, **ramDrv** does handle a **FIODISKFORMAT** request, which always returns OK. All other *ioctl()* requests return an error and set the task's **errno** to **S_ioLib_UNKNOWN_REQUEST**.

INCLUDE FILE **ramDrv.h**

SEE ALSO *dosFsDevInit()*, *dosFsMkfs()*, *rt11FsDevInit()*, *rt11FsMkfs()*, *rawFsDevInit()*, *VxWorks Programmer's Guide: I/O System, Local File Systems*

rawFsLib

NAME **rawFsLib** – raw block device file system library

ROUTINES *rawFsDevInit()* – associate a block device with raw volume functions
rawFsInit() – prepare to use the raw volume library
rawFsModeChange() – modify the mode of a raw device volume
rawFsReadyChange() – notify **rawFsLib** of a change in ready status
rawFsVolUnmount() – disable a raw device volume

DESCRIPTION This library provides basic services for disk devices that do not use a standard file or directory structure. The disk volume is treated much like a large file. Portions of it may be read, written, or the current position within the disk may be changed. However, there is no high-level organization of the disk into files or directories.

USING THIS LIBRARY

The various routines provided by the VxWorks raw "file system" (rawFs) may be separated into three broad groups: general initialization, device initialization, and file

system operation.

The *rawFsInit()* routine is the principal initialization function; it need only be called once, regardless of how many rawFs devices will be used.

A separate rawFs routine is used for device initialization. For each rawFs device, *rawFsDevInit()* must be called to install the device.

Several routines are provided to inform the file system of changes in the system environment. The *rawFsModeChange()* routine may be used to modify the readability or writability of a particular device. The *rawFsReadyChange()* routine is used to inform the file system that a disk may have been swapped and that the next disk operation should first remount the disk. The *rawFsVolUnmount()* routine informs the file system that a particular device should be synchronized and unmounted, generally in preparation for a disk change.

INITIALIZATION

Before any other routines in **rawFsLib** can be used, *rawFsInit()* must be called to initialize the library. This call specifies the maximum number of raw device file descriptors that can be open simultaneously and allocates memory for that many raw file descriptors. Any attempt to open more raw device file descriptors than the specified maximum will result in errors from *open()* or *creat()*.

During the *rawFsInit()* call, the raw device library is installed as a driver in the I/O system driver table. The driver number associated with it is then placed in a global variable, **rawFsDrvNum**.

This initialization is enabled when the configuration macro **INCLUDE_RAWFS** is defined; *rawFsInit()* is then called from the root task, *usrRoot()*, in **usrConfig.c**.

DEFINING A RAW DEVICE

To use this library for a particular device, the device structure used by the device driver must contain, as the very first item, a block device description structure (**BLK_DEV**). This must be initialized before calling *rawFsDevInit()*. In the **BLK_DEV** structure, the driver includes the addresses of five routines it must supply: one that reads one or more blocks, one that writes one or more blocks, one that performs I/O control (*ioctl()*) on the device, one that checks the status of the the device, and one that resets the device. The **BLK_DEV** structure also contains fields that describe the physical configuration of the device. For more information about defining block devices, see the *VxWorks Programmer's Guide: I/O System*.

The *rawFsDevInit()* routine is used to associate a device with the **rawFsLib** functions. The *volName* parameter expected by *rawFsDevInit()* is a pointer to a name string, to be used to identify the device. This will serve as the pathname for I/O operations which operate on the device. This name will appear in the I/O system device table, which may be displayed using *iosDevShow()*.

The *pBlkDev* parameter that *rawFsDevInit()* expects is a pointer to the **BLK_DEV** structure describing the device and contains the addresses of the required driver functions. The syntax of the *rawFsDevInit()* routine is as follows:

```
rawFsDevInit
(
    char    *volName, /* name to be used for volume */
    BLK_DEV *pBlkDev  /* pointer to device descriptor */
)
```

Unlike the VxWorks DOS and RT-11 file systems, raw volumes do not require an **FIODISKINIT** *ioctl()* function to initialize volume structures. (Such an *ioctl()* call can be made for a raw volume, but it has no effect.) As a result, there is no "make file system" routine for raw volumes (for comparison, see the manual entries for *dosFsMkfs()* and *rt11Mkfs()*).

When **rawFsLib** receives a request from the I/O system, after *rawFsDevInit()* has been called, it calls the device driver routines (whose addresses were passed in the **BLK_DEV** structure) to access the device.

MULTIPLE LOGICAL DEVICES

The block number passed to the block read and write routines is an absolute number, starting from block 0 at the beginning of the device. If desired, the driver may add an offset from the beginning of the physical device before the start of the logical device. This would normally be done by keeping an offset parameter in the driver's device-specific structure, and adding the proper number of blocks to the block number passed to the read and write routines. See the **ramDrv** manual entry for an example.

UNMOUNTING VOLUMES (CHANGING DISKS)

A disk should be unmounted before it is removed. When unmounted, any modified data that has not been written to the disk will be written out. A disk may be unmounted by either calling *rawFsVolUnmount()* directly or calling *ioctl()* with a **FIODISKCHANGE** function code.

There may be open file descriptors to a raw device volume when it is unmounted. If this is the case, those file descriptors will be marked as obsolete. Any attempts to use them for further I/O operations will return an **S_rawFsLib_FD_OBSOLETE** error. To free such file descriptors, use the *close()* call, as usual. This will successfully free the descriptor, but will still return **S_rawFsLib_FD_OBSOLETE**.

SYNCHRONIZING VOLUMES

A disk should be "synchronized" before it is unmounted. To synchronize a disk means to write out all buffered data (the write buffers associated with open file descriptors), so that the disk is updated. It may or may not be necessary to explicitly synchronize a disk, depending on how (or if) the driver issues the *rawFsVolUnmount()* call.

When *rawFsVolUnmount()* is called, an attempt will be made to synchronize the device before unmounting. However, if the *rawFsVolUnmount()* call is made by a driver in response to a disk being removed, it is obviously too late to synchronize. Therefore, a separate *ioctl()* call specifying the **FIOSYNC** function should be made before the disk is removed. (This could be done in response to an operator command.)

If the disk will still be present and writable when *rawFsVolUnmount()* is called, it is not necessary to first synchronize the disk. In all other circumstances, failure to synchronize the volume before unmounting may result in lost data.

IOCTL FUNCTIONS The VxWorks raw block device file system supports the following *ioctl()* functions. The functions listed are defined in the header **ioLib.h**.

FIODISKFORMAT

Formats the entire disk with appropriate hardware track and sector marks. No file system is initialized on the disk by this request. Note that this is a driver-provided function:

```
fd = open ("DEV1:", O_WRONLY);  
status = ioctl (fd, FIODISKFORMAT, 0);
```

FIODISKINIT

Initializes a raw file system on the disk volume. Since there are no file system structures, this functions performs no action. It is provided only for compatibility with other VxWorks file systems.

FIODISKCHANGE

Announces a media change. It performs the same function as *rawFsReadyChange()*. This function may be called from interrupt level:

```
status = ioctl (fd, FIODISKCHANGE, 0);
```

FIOUNMOUNT

Unmounts a disk volume. It performs the same function as *rawFsVolUnmount()*. This function must not be called from interrupt level:

```
status = ioctl (fd, FIOUNMOUNT, 0);
```

FIOGETNAME

Gets the file name of the file descriptor and copies it to the buffer *nameBuf*:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

FIOSEEK

Sets the current byte offset on the disk to the position specified by *newOffset*:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

FIOWHERE

Returns the current byte position from the start of the device for the specified file descriptor. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

FIOFLUSH

Writes all modified file descriptor buffers to the physical device.

```
status = ioctl (fd, FIOFLUSH, 0);
```

FIOSYNC

Performs the same function as FIOFLUSH.

FIONREAD

Copies to *unreadCount* the number of bytes from the current file position to the end of the device:

```
status = ioctl (fd, FIONREAD, &unreadCount);
```

INCLUDE FILES **rawFsLib.h**

SEE ALSO **ioLib, iosLib, dosFsLib, rt11FsLib, ramDrv,** *VxWorks Programmer's Guide: I/O System, Local File Systems*

rebootLib

NAME **rebootLib** – reboot support library

ROUTINES *reboot()* – reset network devices and transfer control to boot ROMs
rebootHookAdd() – add a routine to be called at reboot

DESCRIPTION This library provides reboot support. To restart VxWorks, the routine *reboot()* can be called at any time by typing CTRL-X from the shell. Shutdown routines can be added with *rebootHookAdd()*. These are typically used to reset or synchronize hardware. For example, **netLib** adds a reboot hook to cause all network interfaces to be reset. Once the reboot hooks have been run, *sysToMonitor()* is called to transfer control to the boot ROMs. For more information, see the manual entry for bootInit.

DEFICIENCIES The order in which hooks are added is the order in which they are run. As a result, **netLib** will kill the network, and no user-added hook routines will be able to use the network. There is no *rebootHookDelete()* routine.

INCLUDE FILES **rebootLib.h**

SEE ALSO **sysLib, bootConfig, bootInit**

remLib

| | |
|----------------------|---|
| NAME | remLib – remote command library |
| ROUTINES | <i>rcmd()</i> – execute a shell command on a remote machine <i>rresvport()</i> – open a socket with a privileged port bound to it <i>remCurIdGet()</i> – get the current user name and password <i>remCurIdSet()</i> – set the remote user name and password <i>iam()</i> – set the remote user name and password <i>whoami()</i> – display the current remote identity <i>bindresvport()</i> – bind a socket to a privileged IP port |
| DESCRIPTION | This library provides routines to support remote command functions. The <i>rcmd()</i> and <i>rresvport()</i> routines use protocols implemented in UNIX BSD 4.3; they support remote command execution, and the opening of a socket with a bound privileged port, respectively. Other routines in this library authorize network file access via netDrv. |
| INCLUDE FILES | remLib.h |
| SEE ALSO | inetLib , <i>VxWorks Programmer's Guide: Network</i> |

resolvLib

| | |
|--------------------|--|
| NAME | resolvLib – DNS resolver library |
| ROUTINES | <i>resolvInit()</i> – initialize the resolver library <i>resolvGetHostByName()</i> – query the DNS server for the IP address of a host <i>resolvGetHostByAddr()</i> – query the DNS server for the host name of an IP address <i>resolvParamsSet()</i> – set the parameters which control the resolver library <i>resolvParamsGet()</i> – get the parameters which control the resolver library <i>resolvDNExpand()</i> – expand a DNS compressed name from a DNS packet <i>resolvDNComp()</i> – compress a DNS name in a DNS packet <i>resolvQuery()</i> – construct a query, send it, wait for a response <i>resolvMkQuery()</i> – create all types of DNS queries <i>resolvSend()</i> – send a pre-formatted query and return the answer |
| DESCRIPTION | This library provides the client-side services for DNS (Domain Name Service) queries. DNS queries come from applications that require translation of IP addresses to host names and back. If you include this library in VxWorks, it extends the services of the host library. The interface to this library is described in hostLib . The hostLib interface uses |

resolver services to get IP and host names. In addition, the resolver can query multiple DNS servers, if necessary, to add redundancy for queries.

There are two interfaces available for the resolver library. One is a high-level interface suitable for most applications. The other is also a low-level interface for more specialized applications, such as mail protocols.

USING THIS LIBRARY

By default, a VxWorks build does not include the resolver code. In addition, VxWorks is delivered with the resolver library disabled. To include the resolver library in the VxWorks image, edit **config/all/configAll.h** and include the definition:

```
#define INCLUDE_DNS_RESOLVER
```

To enable the resolver services, you need to redefine only one DNS server IP address, changing it from a place-holder value to an actual value. Additional DNS server IP addresses can be configured using *resolvParamsSet()*. To do the initial configuration, edit **configAll.h**, and enter the correct IP address for your domain server in the definition:

```
#define RESOLVER_DOMAIN_SERVER "90.0.0.3"
```

If you do not provide a valid IP address, resolver initialization fails. You also need to configure the domain to which your resolver belongs. To do this, edit **configAll.h** and enter the correct domain name for your organization in the definition:

```
#define RESOLVER_DOMAIN "wrs.com"
```

The last and most important step is to make sure that you have a route to the configured DNS server. If your VxWorks image includes a routing protocol, such as RIP or OSPF, the routes are created for you automatically. Otherwise, you must use *routeAdd()* or *mRouteAdd()* to add the routes to the routing table.

The resolver library comes with a debug option. To turn on debugging, edit **configAll.h** to include the define:

```
#define INCLUDE_DNS_DEBUG
```

This include makes VxWorks print a log of the resolver queries to the console. This feature assumes a single task. Thus, if you are running multiple tasks, your output to the console is a garble of messages from all the tasks.

The resolver library uses UDP to send queries to the DNS server and expects the DNS server to handle recursion. You can change the resolver parameters at any time after the library has been initialized with *resolvInit()*. However, it is strongly recommended that you change parameters only shortly after initialization, or when there are no other tasks accessing the resolver library.

Your procedure for changing any of the resolver parameter should start with a call to *resolvParamsGet()* to retrieve the active parameters. Then you can change the query order (defaults to query DNS server only), the domain name, or add DNS server IP addresses. After the parameters are changed, call *resolvParamsSet()*. For the values you

can use when accessing resolver library services, see the header files **resolvLib.h**, **resolv/resolv.h**, and **resolv/nameser.h**.

INCLUDE FILES **resolvLib.h**

SEE ALSO **hostLib**

ripLib

NAME **ripLib** – Routing Information Protocol (RIP) v1 and v2 library

ROUTINES

- ripLibInit()* – initialize the RIP routing library
- ripRouteShow()* – display the internal routing table maintained by RIP
- ripAuthHookAdd()* – add an authentication hook to a RIP interface
- ripAuthHookDelete()* – remove an authentication hook from a RIP interface
- ripAuthHook()* – sample authentication hook
- ripLeakHookAdd()* – add a hook to bypass the RIP and kernel routing tables
- ripLeakHookDelete()* – remove a table bypass hook from a RIP interface
- ripSendHookAdd()* – add an update filter to a RIP interface
- ripSendHookDelete()* – remove an update filter from a RIP interface
- ripIfSearch()* – add new interfaces to the internal list
- ripIfReset()* – alter the RIP configuration after an interface changes
- ripFilterEnable()* – activate strict border gateway filtering
- ripFilterDisable()* – prevent strict border gateway filtering
- ripShutdown()* – terminate all RIP processing
- ripDebugLevelSet()* – specify amount of debugging output

DESCRIPTION This library implements versions 1 and 2 of the Routing Information Protocol (RIP). The protocol is intended to operate as an interior gateway protocol within a relatively small network with a longest path of 15 hops.

HIGH-LEVEL INTERFACE

The *ripLibInit()* routine links this library into the VxWorks image and begins a RIP session. This happens automatically if **INCLUDE_RIP** is defined at the time the image is built. Once started, RIP will maintain the network routing table until deactivated by a call to the *ripShutdown()* routine, which will remove all route entries and disable the RIP library routines. All RIP requests and responses are handled as defined in the RFC specifications. RFC 1058 defines the basic protocol operation and RFC 1723 details the extensions which implement version 2.

When acting as a supplier, outgoing route updates are filtered using simple split horizon. Split horizon with poisoned reverse is not currently available. Additional route entries may be excluded from the periodic update with the *ripSendHookAdd()* routine.

If a RIP session is terminated, the networking subsystem may not function correctly until RIP is restarted with a new call to *ripLibInit()* unless routing information is provided by some other method.

CONFIGURATION INTERFACE

By default, a RIP session only uses the network interfaces created before it started. The *ripIfSearch()* routine allows RIP to recognize any interfaces added to the system after that point. If the address or netmask of an existing interface is changed during a RIP session, the *ripIfReset()* routine must be used to update the RIP configuration appropriately. The current RIP implementation also automatically performs the border gateway filtering required by the RFC specification. Those restrictions provide correct operation in a mixed environment of RIP-1 and RIP-2 routers. The *ripFilterDisable()* routine will remove those limitations, and may produce more efficient routing for some topologies. That routine must not be used if any version 1 routers are present. The *ripFilterEnable()* routine will restore the default behavior.

AUTHENTICATION INTERFACE

By default, authentication is disabled, but may be activated by an SNMP agent on an interface-specific basis. While authentication is disabled, any RIP-2 messages containing authentication entries are discarded. When enabled, all RIP-2 messages without authentication entries are automatically rejected. To fully support authentication, an authentication routine should be specified with the *ripAuthHookAdd()* routine. The specified function will be called to screen every RIP-1 message and all unverified RIP-2 messages containing authentication entries. It may be removed with the *ripAuthHookDelete()* routine. All RIP-1 and unverified RIP-2 messages will be discarded while authentication is enabled unless a hook is present.

OPTIONAL INTERFACE

The *ripLeakHookAdd()* routine allows the use of an alternative routing protocol which uses RIP as a transport mechanism. The specified function can prevent the RIP session from creating any table entries from the received messages. The *ripLeakHookDelete()* routine will restore the default operation.

DEBUGGING INTERFACE

As required by the RFC specification, the obsolete traceon and traceoff messages are not supported by this implementation. The *ripRouteShow()* routine will display the contents of the internal RIP routing table. Routines such as *mRouteShow()* to display the corresponding kernel routing table will also be available if `INCLUDE_NET_SHOW` is defined when the image is built. If additional information is required, the *ripDebugLevelSet()* routine will enable predefined debugging messages which will be sent to the standard output.

INCLUDE FILES *ripLib.h*

SEE ALSO RFC 1058, RFC 1723

rlogLib

| | |
|----------------------|---|
| NAME | rlogLib – remote login library |
| ROUTINES | <i>rlogInit()</i> – initialize the remote login facility <i>rlogind()</i> – the VxWorks remote login daemon <i>rlogin()</i> – log in to a remote host |
| DESCRIPTION | <p>This library provides a remote login facility for VxWorks that uses the UNIX rlogin protocol (as implemented in UNIX BSD 4.3) to allow users at a VxWorks terminal to log in to remote systems via the network, and users at remote systems to log in to VxWorks via the network.</p> <p>A VxWorks user may log in to any other remote VxWorks or UNIX system via the network by calling <i>rlogin()</i> from the shell.</p> <p>The remote login daemon, <i>rlogind()</i>, allows remote users to log in to VxWorks. The daemon is started by calling <i>rlogInit()</i>, which is called automatically when the configuration macro INCLUDE_RLOGIN is defined. The remote login daemon accepts remote login requests from another VxWorks or UNIX system, and causes the shell's input and output to be redirected to the remote user.</p> <p>Internally, <i>rlogind()</i> provides a tty-like interface to the remote user through the use of the VxWorks pseudo-terminal driver <code>ptyDrv</code>.</p> |
| INCLUDE FILES | rlogLib.h |
| SEE ALSO | ptyDrv , telnetLib , UNIX BSD 4.3 manual entries for rlogin , rlogind , and pty |

rngLib

| | |
|-----------------|---|
| NAME | rngLib – ring buffer subroutine library |
| ROUTINES | <i>rngCreate()</i> – create an empty ring buffer <i>rngDelete()</i> – delete a ring buffer <i>rngFlush()</i> – make a ring buffer empty <i>rngBufGet()</i> – get characters from a ring buffer <i>rngBufPut()</i> – put bytes into a ring buffer <i>rngIsEmpty()</i> – test if a ring buffer is empty <i>rngIsFull()</i> – test if a ring buffer is full (no more room) <i>rngFreeBytes()</i> – determine the number of free bytes in a ring buffer <i>rngNBytes()</i> – determine the number of bytes in a ring buffer |

rngPutAhead() – put a byte ahead in a ring buffer without moving ring pointers
rngMoveAhead() – advance a ring pointer by *n* bytes

DESCRIPTION

This library provides routines for creating and using ring buffers, which are first-in-first-out circular buffers. The routines simply manipulate the ring buffer data structure; no kernel functions are invoked. In particular, ring buffers by themselves provide no task synchronization or mutual exclusion.

However, the ring buffer pointers are manipulated in such a way that a reader task (invoking *rngBufGet()*) and a writer task (invoking *rngBufPut()*) can access a ring simultaneously without requiring mutual exclusion. This is because readers only affect a *read* pointer and writers only affect a *write* pointer in a ring buffer data structure. However, access by multiple readers or writers *must* be interlocked through a mutual exclusion mechanism (i.e., a mutual-exclusion semaphore guarding a ring buffer).

This library also supplies two macros, **RNG_ELEM_PUT** and **RNG_ELEM_GET**, for putting and getting single bytes from a ring buffer. They are defined in **rngLib.h**.

```
int RNG_ELEM_GET (ringId, pch, fromP)
int RNG_ELEM_PUT (ringId, ch, toP)
```

Both macros require a temporary variable *fromP* or *toP*, which should be declared as **register int** for maximum efficiency. **RNG_ELEM_GET** returns 1 if there was a character available in the buffer; it returns 0 otherwise. **RNG_ELEM_PUT** returns 1 if there was room in the buffer; it returns 0 otherwise. These are somewhat faster than *rngBufPut()* and *rngBufGet()*, which can put and get multi-byte buffers.

INCLUDE FILES

rngLib.h

routeLib

NAME

routeLib – network route manipulation library

ROUTINES

routeAdd() – add a route
routeNetAdd() – add a route to a destination that is a network
routeDelete() – delete a route
mRouteAdd() – add multiple routes to the same destination
mRouteEntryAdd() – add a protocol-specific route to the routing table
mRouteEntryDelete() – delete route from the routing table
mRouteDelete() – delete a route from the routing table
routeProtoPrioritySet() – set the priority of routes added by the routing protocol

DESCRIPTION

This library contains the routines for inspecting the routing table, as well as routines for adding and deleting routes from that table. If you do not configure VxWorks to include a

routing protocol, such as RIP or OSPF, you can use these routines to maintain the routing tables manually.

INCLUDE FILES **routeLib.h**

SEE ALSO **hostLib**, *Network Programmer's Guide*

rpcLib

NAME **rpcLib** – Remote Procedure Call (RPC) support library

ROUTINES **rpcInit()** – initialize the RPC package
 rpcTaskInit() – initialize a task's access to the RPC package

DESCRIPTION This library supports Sun Microsystems' Remote Procedure Call (RPC) facility. RPC provides facilities for implementing distributed client/server-based architectures. The underlying communication mechanism can be completely hidden, permitting applications to be written without any reference to network sockets. The package is structured such that lower-level routines can optionally be accessed, allowing greater control of the communication protocols.

For more information and a tutorial on RPC, see Sun Microsystems' *Remote Procedure Call Programming Guide*. For an example of RPC usage, see **/target/unsupported/demo/sprites**.

The RPC facility is enabled when the configuration macro **INCLUDE_RPC** is defined.

VxWorks supports Network File System (NFS), which is built on top of RPC. If NFS is configured into the VxWorks system, RPC is automatically included as well.

IMPLEMENTATION A task must call **rpcTaskInit()** before making any calls to other routines in the RPC library. This routine creates task-specific data structures required by RPC. These task-specific data structures are automatically deleted when the task exits.

Because each task has its own RPC context, RPC-related objects (such as SVCXPRTs and CLIENTs) cannot be shared among tasks; objects created by one task cannot be passed to another for use. Such additional objects must be explicitly deleted (for example, using task deletion hooks).

INCLUDE FILES **rpc.h**

SEE ALSO **nfsLib**, **nfsDrv**, Sun Microsystems' *Remote Procedure Call Programming Guide*

rt11FsLib

| | |
|--------------------|--|
| NAME | rt11FsLib – RT-11 media-compatible file system library |
| ROUTINES | <i>rt11FsDevInit()</i> – initialize the rt11Fs device descriptor <i>rt11FsInit()</i> – prepare to use the rt11Fs library <i>rt11FsMkfs()</i> – initialize a device and create an rt11Fs file system <i>rt11FsDateSet()</i> – set the rt11Fs file system date <i>rt11FsReadyChange()</i> – notify rt11Fs of a change in ready status <i>rt11FsModeChange()</i> – modify the mode of an rt11Fs volume |
| DESCRIPTION | This library provides services for file-oriented device drivers which use the RT-11 file standard. This module takes care of all the necessary buffering, directory maintenance, and RT-11-specific details. |

USING THIS LIBRARY

The various routines provided by the VxWorks RT-11 file system (rt11Fs) may be separated into three broad groups: general initialization, device initialization, and file system operation.

The *rt11FsInit()* routine is the principal initialization function; it need only be called once, regardless of how many rt11Fs devices will be used.

Other rt11Fs routines are used for device initialization. For each rt11Fs device, either *rt11FsDevInit()* or *rt11FsMkfs()* must be called to install the device and define its configuration.

Several functions are provided to inform the file system of changes in the system environment. The *rt11FsDateSet()* routine is used to set the date. The *rt11FsModeChange()* routine is used to modify the readability or writability of a particular device. The *rt11FsReadyChange()* routine is used to inform the file system that a disk may have been swapped, and that the next disk operation should first remount the disk.

INITIALIZING RT11FSLIB

Before any other routines in **rt11FsLib** can be used, *rt11FsInit()* must be called to initialize this library. This call specifies the maximum number of rt11Fs files that can be open simultaneously and allocates memory for that many rt11Fs file descriptors. Attempts to open more files than the specified maximum will result in errors from *open()* or *creat()*.

This initialization is enabled when the configuration macro **INCLUDE_RT11FS** is defined.

DEFINING AN RT-11 DEVICE

To use this library for a particular device, the device structure must contain, as the very first item, a **BLK_DEV** structure. This must be initialized before calling *rt11FsDevInit()*.

In the **BLK_DEV** structure, the driver includes the addresses of five routines which it must supply: one that reads one or more sectors, one that writes one or more sectors, one that performs I/O control on the device (using *ioctl()*), one that checks the status of the device, and one that resets the device. This structure also specifies various physical aspects of the device (e.g., number of sectors, sectors per track, whether the media is removable). For more information about defining block devices, see the *VxWorks Programmer's Guide: I/O System*.

The device is associated with the **rt11Fs** file system by the *rt11FsDevInit()* call. The arguments to *rt11FsDevInit()* include the name to be used for the **rt11Fs** volume, a pointer to the **BLK_DEV** structure, whether the device uses RT-11 standard skew and interleave, and the maximum number of files that can be contained in the device directory.

Thereafter, when the file system receives a request from the I/O system, it simply calls the provided routines in the device driver to fulfill the request.

RTFMT

The RT-11 standard defines a peculiar software interleave and track-to-track skew as part of the format. The *rtFmt* parameter passed to *rt11FsDevInit()* should be TRUE if this formatting is desired. This should be the case if strict RT-11 compatibility is desired, or if files must be transferred between the development and target machines using the VxWorks-supplied RT-11 tools. Software interleave and skew will automatically be dealt with by **rt11FsLib**.

When *rtFmt* has been passed as TRUE and the maximum number of files is specified **RT_FILES_FOR_2_BLOCK_SEG**, the driver does not need to do anything else to maintain RT-11 compatibility (except to add the track offset as described above).

Note that if the number of files specified is different than **RT_FILES_FOR_2_BLOCK_SEG** under either a VxWorks system or an RT-11 system, compatibility is lost because VxWorks allocates a contiguous directory, whereas RT-11 systems create chained directories.

MULTIPLE LOGICAL DEVICES AND RT-11 COMPATIBILITY

The sector number passed to the sector read and write routines is an absolute number, starting from sector 0 at the beginning of the device. If desired, the driver may add an offset from the beginning of the physical device before the start of the logical device. This would normally be done by keeping an offset parameter in the device-specific structure of the driver, and adding the proper number of sectors to the sector number passed to the read and write routines.

The RT-11 standard defines the disk to start on track 1. Track 0 is set aside for boot information. Therefore, in order to retain true compatibility with RT-11 systems, a one-track offset (i.e., the number of sectors in one track) needs to be added to the sector numbers passed to the sector read and write routines, and the device size needs to be declared as one track smaller than it actually is. This must be done by the driver using **rt11FsLib**; the library does not add such an offset automatically.

In the VxWorks RT-11 implementation, the directory is a fixed size, able to contain at least as many files as specified in the call to *rt11FsDevInit()*. If the maximum number of files is specified to be `RT_FILES_FOR_2_BLOCK_SEG`, strict RT-11 compatibility is maintained, because this is the initial allocation in the RT-11 standard.

RT-11 FILE NAMES File names in the RT-11 file system use six characters, followed by a period (`.`), followed by an optional three-character extension.

DIRECTORY ENTRIES

An *ioctl()* call with the `FIODIRENTRY` function returns information about a particular directory entry. A pointer to a `REQ_DIR_ENTRY` structure is passed as the parameter. The field `entryNum` in the `REQ_DIR_ENTRY` structure must be set to the desired entry number. The name of the file, its size (in bytes), and its creation date are returned in the structure. If the specified entry is empty (i.e., if it represents an unallocated section of the disk), the name will be an empty string, the size will be the size of the available disk section, and the date will be meaningless. Typically, entries are accessed sequentially, starting with `entryNum = 0`, until the terminating entry is reached, indicated by a return of `ERROR`.

DIRECTORIES IN MEMORY

A copy of the directory for each volume is kept in memory (in the `RT_VOL_DESC` structure). This speeds up directory accesses, but requires that `rt11FsLib` be notified when disks are changed (i.e., floppies are swapped). If the driver can find this out (by interrogating controller status or by receiving an interrupt), the driver simply calls *rt11FsReadyChange()* when a disk is inserted or removed. The library `rt11FsLib` will automatically try to remount the device next time it needs it.

If the driver does not have access to the information that disk volumes have been changed, the *changeNoWarn* parameter should be set to `TRUE` when the device is defined using *rt11FsDevInit()*. This will cause the disk to be automatically remounted before each *open()*, *creat()*, *delete()*, and directory listing.

The routine *rt11FsReadyChange()* can also be called by user tasks, by issuing an *ioctl()* call with `FIODISKCHANGE` as the function code.

ACCESSING THE RAW DISK

As a special case in *open()* and *creat()* calls, `rt11FsLib` recognizes a `NULL` file name to indicate access to the entire "raw" disk, as opposed to a file on the disk. Access in raw mode is useful for a disk that has no file system. For example, to initialize a new file system on the disk, use an *ioctl()* call with `FIODISKINIT`. To read the directory of a disk for which no file names are known, open the raw disk and use an *ioctl()* call with the function `FIODIRENTRY`.

HINTS

The RT-11 file system is much simpler than the more common UNIX or MS-DOS file systems. The advantage of RT-11 is its speed; file access is made in at most one seek because all files are contiguous. Some of the most common errors for users with a UNIX background are:

- Only a single create at a time may be active per device.
- File size is set by the first create and close sequence; use *lseek()* to ensure a specific file size; there is no append function to expand a file.
- Files are strictly block oriented; unused portions of a block are filled with NULLs -- there is no end-of-file marker other than the last block.

IOCTL FUNCTIONS The rt11Fs file system supports the following *ioctl()* functions. The functions listed are defined in the header **ioLib.h**. Unless stated otherwise, the file descriptor used for these functions can be any file descriptor open to a file or to the volume itself.

FIODISKFORMAT

Formats the entire disk with appropriate hardware track and sector marks. No file system is initialized on the disk by this request. Note that this is a driver-provided function:

```
fd = open ("DEV1:", O_WRONLY);  
status = ioctl (fd, FIODISKFORMAT, 0);
```

FIODISKINIT

Initializes an rt11Fs file system on the disk volume. This routine does not format the disk; formatting must be done by the driver. The file descriptor should be obtained by opening the entire volume in raw mode:

```
fd = open ("DEV1:", O_WRONLY);  
status = ioctl (fd, FIODISKINIT, 0);
```

FIODISKCHANGE

Announces a media change. It performs the same function as *rt11FsReadyChange()*. This function may be called from interrupt level:

```
status = ioctl (fd, FIODISKCHANGE, 0);
```

FIOGETNAME

Gets the file name of the file descriptor and copies it to the buffer *nameBuf*:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

FIORENAME

Renames the file to the string *newname*:

```
status = ioctl (fd, FIORENAME, "newname");
```

FIONREAD

Copies to *unreadCount* the number of unread bytes in the file:

```
status = ioctl (fd, FIONREAD, &unreadCount);
```

FIOFLUSH

Flushes the file output buffer. It guarantees that any output that has been requested is actually written to the device.

```
status = ioctl (fd, FIOFLUSH, 0);
```

FIOSEEK

Sets the current byte offset in the file to the position specified by *newOffset*:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

FIOWHERE

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

FIOSQUEEZE

Coalesces fragmented free space on an rt11Fs volume:

```
status = ioctl (fd, FIOSQUEEZE, 0);
```

FIODIRENTRY

Copies information about the specified directory entries to a **REQ_DIR_ENTRY** structure that is defined in **ioLib.h**. The argument *reqis* a pointer to a **REQ_DIR_ENTRY** structure. On entry, the structure contains the number of the directory entry for which information is requested. On return, the structure contains the information on the requested entry. For example, after the following:

```
REQ_DIR_ENTRY req;
req.entryNum = 0;
status = ioctl (fd, FIODIRENTRY, &req);
```

The request structure contains the name, size, and creation date of the file in the first entry (0) of the directory.

FIOREADDIR

Reads the next directory entry. The argument *dirStruct* is a DIR directory descriptor. Normally, *readdir()* is used to read a directory, rather than using the **FIOREADDIR** function directly. See **dirLib**.

```
DIR dirStruct;
fd = open ("directory", O_RDONLY);
status = ioctl (fd, FIOREADDIR, &dirStruct);
```

FIOFSTATGET

Gets file status information (directory entry data). The argument *statStruct* is a pointer to a stat structure that is filled with data describing the specified file. Normally, the *stat()* or *fstat()* routine is used to obtain file information, rather than using the **FIOFSTATGET** function directly. See **dirLib**.

```
struct stat statStruct;
fd = open ("file", O_RDONLY);
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

Any other *ioctl()* function codes are passed to the block device driver for handling.

INCLUDE FILES **rt11FsLib.h**

SEE ALSO **ioLib, iosLib, ramDrv, VxWorks Programmer's Guide: I/O System, Local File Systems**

sa1100Sio

| | |
|------------------------|--|
| NAME | sa1100Sio – Digital Semiconductor SA-1100 UART tty driver |
| ROUTINES | <i>sa1100DevInit()</i> – initialise an SA1100 channel <i>sa1100Int()</i> – handle an interrupt |
| DESCRIPTION | <p>This is the device driver for the Digital Semiconductor SA-1100 UARTs. This chip contains 5 serial ports, but only ports 1 and 3 are usable as UARTs, the others support Universal Serial Bus (USB), SDLC, IrDA Infrared Communications Port (ICP) and Multimedia Communications Port (MCP)/Synchronous Serial Port (SSP).</p> <p>The UARTs are identical in design. They contain a universal asynchronous receiver/transmitter, and a baud-rate generator, The UARTs contain an 8-entry, 8-bit FIFO to buffer outgoing data and a 12-entry 11-bit FIFO to buffer incoming data. If a framing, overrun or parity error occurs during reception, the appropriate error bits are stored in the receive FIFO along with the received data. The only mode of operation supported is with the FIFOs enabled.</p> <p>The UART design does not support modem control input or output signals e.g. DTR, RI, RTS, DCD, CTS and DSR.</p> <p>An interrupt is generated when a framing, parity or receiver overrun error is present within the bottom four entries of the receive FIFO, when the transmit FIFO is half-empty or receive FIFO is one- to two-thirds full, when a begin and end of break is detected on the receiver, and when the receive FIFO is partially full and the receiver is idle for three or more frame periods.</p> <p>Only asynchronous serial operation is supported by the UARTs which supports 7 or 8 bit word lengths with or without parity and with one or two stop bits. The only serial word format supported by the driver is 8 data bits, 1 stop bit, no parity, The default baud rate is determined by the BSP by filling in the SA1100_CHAN structure before calling <i>sa1100DevInit()</i>.</p> <p>The UART supports baud rates from 56.24 to 230.4 kbps.</p> |
| DATA STRUCTURES | An SA1100_CHAN data structure is used to describe each channel, this structure is described in h/drv/sio/sa1100Sio.h . |
| CALLBACKS | Servicing a "transmitter ready" interrupt involves making a callback to a higher level library in order to get a character to transmit. By default, this driver installs dummy callback routines which do nothing. A higher layer library that wants to use this driver (e.g. ttYDrv) will install its own callback routine using the SIO_INSTALL_CALLBACK ioctl command. Likewise, a receiver interrupt handler makes a callback to pass the character to the higher layer library. |

MODES This driver supports both polled and interrupt modes.

USAGE The driver is typically only called by the BSP. The directly callable routines in this module are *sa1100DevInit()*, and *sa1100Int()*.

The BSP's *sysHwInit()* routine typically calls *sysSerialHwInit()*, which initialises the hardware-specific fields in the SA1100_CHAN structure (e.g. register I/O addresses etc) before calling *sa1100DevInit()* which resets the device and installs the driver function pointers. After this the UART will be enabled and ready to generate interrupts, but those interrupts will be disabled in the interrupt controller.

The following example shows the first parts of the initialization:

```
#include "drv/sio/sa1100Sio.h"
LOCAL SA1100_CHAN sa1100Chan[N_SA1100_UART_CHANS];
void sysSerialHwInit (void)
{
    int i;
    for (i = 0; i < N_SA1100_UART_CHANNELS; i++)
    {
        sa1100Chan[i].regs = devParas[i].baseAdrs;
        sa1100Chan[i].baudRate = CONSOLE_BAUD_RATE;
        sa1100Chan[i].xtal = UART_XTAL_FREQ;
        sa1100Chan[i].level = devParas[i].intLevel;
        /* set up GPIO pins and UART pin reassignment */
        ...
        /*
         * Initialise driver functions, getTxChar, putRcvChar
         * and channelMode and initialise UART
         */
        sa1100DevInit(&sa1100Chan[i]);
    }
}
```

The BSP's *sysHwInit2()* routine typically calls *sysSerialHwInit2()*, which connects the chip's interrupts via *intConnect()* and enables those interrupts, as in the following:

```
void sysSerialHwInit2 (void)
{
    int i;
    for (i = 0; i < N_SA1100_UART_CHANNELS; i++)
    {
        /* connect and enable interrupts */
        (void)intConnect (INUM_TO_IVEC(devParas[i].vector),
                        sa1100Int, (int) &sa1100Chan[i]);
        intEnable (devParas[i].intLevel);
    }
}
```

saIoLib

- BSP** By convention all the BSP-specific serial initialization is performed in a file called **sysSerial.c**, which is #include'd by **sysLib.c**. **sysSerial.c** implements at least four functions, *sysSerialHwInit()*, *sysSerialHwInit2()*, *sysSerialChanGet()*, and *sysSerialReset()*. The first two have been described above, the others work as follows:
- sysSerialChanGet* is called by *usrRoot* to get the serial channel descriptor associated with a serial channel number. The routine takes a single parameter which is a channel number ranging between zero and **NUM_TTY**. It returns a pointer to the corresponding channel descriptor, **SIO_CHAN ***, which is just the address of the **SA1100_CHAN** structure.
- sysSerialReset* is called from *sysToMonitor()* and should reset the serial devices to an inactive state (prevent them from generating any interrupts).
- INCLUDE FILES** **drv/sio/sa1100Sio.h** **sioLib.h**
- SEE ALSO** *Digital StrongARM SA-1100 Portable Communications Microcontroller, Data Sheet*, *Digital Semiconductor StrongARM SA-1100 Microprocessor Evaluation Platform, User's Guide*

saIoLib

- NAME** **saIoLib** – default transport routines for SNMP subagent
- ROUTINES**
- snmpSaInit()* – initialize the subagent
 - saIoWrite()* – send a packet to the master agent's message queue
 - saIpcFree()* – free the specified IPC mechanism
 - saMsgBuild()* – build and encode a message and send it to the master agent
 - hdrBlkBuild()* – create the header block and the demuxer information
 - envoy_now()* – return the number of clock ticks elapsed since the timer was set
 - envoy_call_timer()* – execute the specified function when the timer expires
- DESCRIPTION**
- This library implements the subagent side of the IPC mechanism used to pass messages between the SNMP master agent and its subagents. In the shipped version of this library, the IPC mechanism is a message queue. However, it is a relatively simple matter to replace the message queue with a socket if you cannot use message queues.
- To set up the IPC mechanism and spawn a task to monitor it, call *snmpSaInit()*. To send a message to the master agent, you can call *saIoWrite()*. However, you will likely never call this function directly. Instead, you will call *hdrBlkBuild()*. Internally, *hdrBlkBuild()* calls *saMsgBuild()*, which calls *snmpSubEncode()* and finally *saIoWrite()*.
- The first message you will transmit using *hdrBlkBuild()* will be a registration message that registers objects and instances as a group in the master agent's MIB tree. If successful, the response to this message will contain a group ID. Make sure that you store this ID so that you can later remove the group from the MIB tree when you want to

deregister the subagent. You also need this ID if you want to register instances of the object just registered.

Exactly how and when you register a subagent is up to you, but keep in mind that you can do so only after the master agent is up and running.

SEE ALSO **saIoLib**

schedPxLib

NAME **schedPxLib** – scheduling library (POSIX)

ROUTINES *sched_setparam()* – set a task’s priority (POSIX)
sched_getparam() – get the scheduling parameters for a specified task (POSIX)
sched_setscheduler() – set scheduling policy and scheduling parameters (POSIX)
sched_getscheduler() – get the current scheduling policy (POSIX)
sched_yield() – relinquish the CPU (POSIX)
sched_get_priority_max() – get the maximum priority (POSIX)
sched_get_priority_min() – get the minimum priority (POSIX)
sched_rr_get_interval() – get the current time slice (POSIX)

DESCRIPTION This library provides POSIX-compliance scheduling routines. The routines in this library allow the user to get and set priorities and scheduling schemes, get maximum and minimum priority values, and get the time slice if round-robin scheduling is enabled.

The POSIX standard specifies a priority numbering scheme in which higher priorities are indicated by larger numbers. The VxWorks native numbering scheme is the reverse of this, with higher priorities indicated by smaller numbers. For example, in the VxWorks native priority numbering scheme, the highest priority task has a priority of 0.

In VxWorks, POSIX scheduling interfaces are implemented using the POSIX priority numbering scheme. This means that the priority numbers used by this library *do not* match those reported and used in all the other VxWorks components. It is possible to change the priority numbering scheme used by this library by setting the global variable **posixPriorityNumbering**. If this variable is set to **FALSE**, the VxWorks native numbering scheme (small number = high priority) is used, and priority numbers used by this library will match those used by the other portions of VxWorks.

The routines in this library are compliant with POSIX 1003.1b. In particular, task priorities are set and reported through the structure **sched_setparam**, which has a single member:

```
struct sched_param            /* Scheduling parameter structure */
{
    int sched_priority;        /* scheduling priority */
};
```

scsi1Lib

POSIX 1003.1b specifies this indirection to permit future extensions through the same calling interface. For example, because *sched_setparam()* takes this structure as an argument (rather than using the priority value directly) its type signature need not change if future schedulers require other parameters.

INCLUDE FILES **sched.h**

SEE ALSO POSIX 1003.1b document, **taskLib**

scsi1Lib

NAME **scsi1Lib** – Small Computer System Interface (SCSI) library (SCSI-1)

ROUTINES No Callable Routines

DESCRIPTION This library implements the Small Computer System Interface (SCSI) protocol in a controller-independent manner. It implements only the SCSI initiator function; the library does not support a VxWorks target acting as a SCSI target. Furthermore, in the current implementation, a VxWorks target is assumed to be the only initiator on the SCSI bus, although there may be multiple targets (SCSI peripherals) on the bus.

The implementation is transaction based. A transaction is defined as the selection of a SCSI device by the initiator, the issuance of a SCSI command, and the sequence of data, status, and message phases necessary to perform the command. A transaction normally completes with a "Command Complete" message from the target, followed by disconnection from the SCSI bus. If the status from the target is "Check Condition," the transaction continues; the initiator issues a "Request Sense" command to gain more information on the exception condition reported.

Many of the subroutines in **scsi1Lib** facilitate the transaction of frequently used SCSI commands. Individual command fields are passed as arguments from which SCSI Command Descriptor Blocks are constructed, and fields of a **SCSI_TRANSACTION** structure are filled in appropriately. This structure, along with the **SCSI_PHYS_DEV** structure associated with the target SCSI device, is passed to the routine whose address is indicated by the **scsiTransact** field of the **SCSI_CTRL** structure associated with the relevant SCSI controller.

The function variable **scsiTransact** is set by the individual SCSI controller driver. For off-board SCSI controllers, this routine rearranges the fields of the **SCSI_TRANSACTION** structure into the appropriate structure for the specified hardware, which then carries out the transaction through firmware control. Drivers for an on-board SCSI-controller chip can use the *scsiTransact()* routine in **scsiLib** (which invokes the *scsi1Transact()* routine in **scsi1Lib**), as long as they provide the other functions specified in the **SCSI_CTRL** structure.

Note that no disconnect/reconnect capability is currently supported.

SUPPORTED SCSI DEVICES

The `scsi1Lib` library supports use of SCSI peripherals conforming to the standards specified in *Common Command Set (CCS) of the SCSI, Rev. 4.B*. Most SCSI peripherals currently offered support CCS. While an attempt has been made to have `scsi1Lib` support non-CCS peripherals, not all commands or features of this library are guaranteed to work with them. For example, auto-configuration may be impossible with non-CCS devices, if they do not support the INQUIRY command.

Not all classes of SCSI devices are supported. However, the `scsiLib` library provides the capability to transact any SCSI command on any SCSI device through the `FIOSCSICOMMAND` function of the `scsiIoctl()` routine.

Only direct-access devices (disks) are supported by a file system. For other devices, additional higher-level software is necessary to map user commands to SCSI transactions.

CONFIGURING SCSI CONTROLLERS

The routines to create and initialize a specific SCSI controller are particular to the controller and normally are found in its library module. The normal calling sequence is:

```
xxCtrlCreate (...); /* parameters are controller specific */
xxCtrlInit (...); /* parameters are controller specific */
```

The conceptual difference between the two routines is that `xxCtrlCreate()` alloc's memory for the `xx SCSI_CTRL` data structure and initializes information that is never expected to change (for example, clock rate). The remaining fields in the `xx SCSI_CTRL` structure are initialized by `xxCtrlInit()` and any necessary registers are written on the SCSI controller to effect the desired initialization. This routine can be called multiple times, although this is rarely required. For example, the bus ID of the SCSI controller can be changed without rebooting the VxWorks system.

CONFIGURING PHYSICAL SCSI DEVICES

Before a device can be used, it must be "created," that is, declared. This is done with `scsiPhysDevCreate()` and can only be done after a `SCSI_CTRL` structure exists and has been properly initialized.

```
SCSI_PHYS_DEV *scsiPhysDevCreate
(
    SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
    int devBusId,          /* device's SCSI bus ID */
    int devLUN,            /* device's logical unit number */
    int reqSenseLength,    /* length of REQUEST SENSE data dev returns */
    int devType,           /* type of SCSI device */
    BOOL removable,        /* whether medium is removable */
    int numBlocks,         /* number of blocks on device */
    int blockSize          /* size of a block in bytes */
)
```

scsi1Lib

Several of these parameters can be left unspecified, as follows:

reqSenseLength

If 0, issue a **REQUEST_SENSE** to determine a request sense length.

devType

If -1, issue an **INQUIRY** to determine the device type.

numBlocks, blockSize

If 0, issue a **READ_CAPACITY** to determine the number of blocks.

The above values are recommended, unless the device does not support the required commands, or other non-standard conditions prevail.

LOGICAL PARTITIONS ON BLOCK DEVICES

It is possible to have more than one logical partition on a SCSI block device. This capability is currently not supported for removable media devices. A partition is an array of contiguously addressed blocks with a specified starting block address and a specified number of blocks. The *scsiBlkDevCreate()* routine is called once for each block device partition. Under normal usage, logical partitions should not overlap.

```
SCSI_BLK_DEV *scsiBlkDevCreate
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device info */
    int             numBlocks,     /* number of blocks in block device */
    int             blockOffset    /* address of first block in volume */
)
```

Note that if *numBlocks* is 0, the rest of the device is used.

ATTACHING FILE SYSTEMS TO LOGICAL PARTITIONS

Files cannot be read or written to a disk partition until a file system (such as dosFs or rt11Fs) has been initialized on the partition. For more information, see the documentation in **dosFsLib** or **rt11FsLib**.

TRANSMITTING ARBITRARY COMMANDS TO SCSI DEVICES

The **scsi1Lib** library provides routines that implement many common SCSI commands. Still, there are situations that require commands that are not supported by **scsi1Lib** (for example, writing software to control non-direct access devices). Arbitrary commands are handled with the **FIOCSICOMMAND** option to *scsiIoctl()*. The *arg* parameter for **FIOCSICOMMAND** is a pointer to a valid **SCSI_TRANSACTION** structure. Typically, a call to *scsiIoctl()* is written as a subroutine of the form:

```
STATUS myScsiCommand
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    char *         buffer,        /* ptr to data buffer */
    int            bufLength,     /* length of buffer in bytes */
    int            someParam     /* param. specifiable in cmd block */
)
```

```

)
{
SCSI_COMMAND myScsiCmdBlock;      /* SCSI command byte array */
SCSI_TRANSACTION myScsiXaction;    /* info on a SCSI transaction */
/* fill in fields of SCSI_COMMAND structure */
myScsiCmdBlock [0] = MY_COMMAND_OPCODE; /* the required opcode */
.
myScsiCmdBlock [X] = (UINT8) someParam; /* for example */
.
myScsiCmdBlock [N-1] = MY_CONTROL_BYTE; /* typically == 0 */
/* fill in fields of SCSI_TRANSACTION structure */
myScsiXaction.cmdAddress    = myScsiCmdBlock;
myScsiXaction.cmdLength    = <# of valid bytes in myScsiCmdBlock>;
myScsiXaction.dataAddress  = (UINT8 *) buffer;
myScsiXaction.dataDirection = <O_RDONLY (0) or O_WRONLY (1)>;
myScsiXaction.dataLength   = bufLength;
myScsiXaction.cmdTimeout   = timeout in usec;
/* if dataDirection is O_RDONLY, and the length of the input data is
 * variable, the following parameter specifies the byte # (min == 0)
 * of the input data which will specify the additional number of
 * bytes available
 */
myScsiXaction.addLengthByte = X;
if (scsiIoctl (pScsiPhysDev, FIOSCSICOMMAND, &myScsiXaction) == OK)
    return (OK);
else
    /* optionally perform retry or other action based on value of
     * myScsiXaction.statusByte
     */
    return (ERROR);
}

```

INCLUDE FILES scsiLib.h, scsi1Lib.h

SEE ALSO **dosFsLib, rt11FsLib**, *American National Standards for Information Systems – Small Computer System Interface (SCSI)*, ANSI X3.131-1986, *VxWorks Programmer's Guide: I/O System, Local File Systems*

scsi2Lib

| | |
|--------------------|---|
| NAME | scsi2Lib – Small Computer System Interface (SCSI) library (SCSI-2) |
| ROUTINES | <p><i>scsi2IfInit()</i> – initialize the SCSI-2 interface to scsiLib <i>scsiTargetOptionsSet()</i> – set options for one or all SCSI targets <i>scsiTargetOptionsGet()</i> – get options for one or all SCSI targets <i>scsiPhysDevShow()</i> – show status information for a physical device <i>scsiCacheSynchronize()</i> – synchronize the caches for data coherency <i>scsiIdentMsgBuild()</i> – build an identification message <i>scsiIdentMsgParse()</i> – parse an identification message <i>scsiMsgOutComplete()</i> – perform post-processing after a SCSI message is sent <i>scsiMsgOutReject()</i> – perform post-processing when an outgoing message is rejected <i>scsiMsgInComplete()</i> – handle a complete SCSI message received from the target <i>scsiSyncXferNegotiate()</i> – initiate or continue negotiating transfer parameters <i>scsiWideXferNegotiate()</i> – initiate or continue negotiating wide parameters <i>scsiThreadInit()</i> – perform generic SCSI thread initialization <i>scsiCacheSnoopEnable()</i> – inform SCSI that hardware snooping of caches is enabled <i>scsiCacheSnoopDisable()</i> – inform SCSI that hardware snooping of caches is disabled</p> |
| DESCRIPTION | <p>This library implements the Small Computer System Interface (SCSI) protocol in a controller-independent manner. It implements only the SCSI initiator function as defined in the SCSI-2 ANSI specification. This library does not support a VxWorks target acting as a SCSI target.</p> <p>The implementation is transaction based. A transaction is defined as the selection of a SCSI device by the initiator, the issuance of a SCSI command, and the sequence of data, status, and message phases necessary to perform the command. A transaction normally completes with a "Command Complete" message from the target, followed by disconnection from the SCSI bus. If the status from the target is "Check Condition," the transaction continues; the initiator issues a "Request Sense" command to gain more information on the exception condition reported.</p> <p>Many of the subroutines in scsi2Lib facilitate the transaction of frequently used SCSI commands. Individual command fields are passed as arguments from which SCSI Command Descriptor Blocks are constructed, and fields of a SCSI_TRANSACTION structure are filled in appropriately. This structure, along with the SCSI_PHYS_DEV structure associated with the target SCSI device, is passed to the routine whose address is indicated by the scsiTransact field of the SCSI_CTRL structure associated with the relevant SCSI controller. The above mentioned structures are defined in scsi2Lib.h.</p> <p>The function variable scsiTransact is set by the individual SCSI controller driver. For off-board SCSI controllers, this routine rearranges the fields of the SCSI_TRANSACTION structure into the appropriate structure for the specified hardware, which then carries out the transaction through firmware control. Drivers for an on-board SCSI-controller chip</p> |

can use the *scsiTransact()* routine in **scsiLib** (which invokes the *scsi2Transact()* routine in **scsi2Lib**), as long as they provide the other functions specified in the **SCSI_CTRL** structure.

SCSI TRANSACTION TIMEOUT

Associated with each transaction is a time limit (specified in microseconds, but measured with the resolution of the system clock). If the transaction has not completed within this time limit, the SCSI library aborts it; the called routine fails with a corresponding error code. The timeout period includes time spent waiting for the target device to become free to accept the command.

The semantics of the timeout should guarantee that the caller waits no longer than the transaction timeout period, but in practice this may depend on the state of the SCSI bus and the connected target device when the timeout occurs. If the target behaves correctly according to the SCSI specification, proper timeout behavior results. However, in certain unusual cases--for example, when the target does not respond to an asserted ATN signal--the caller may remain blocked for longer than the timeout period.

If the transaction timeout causes problems in your system, you can set the value of either or both the global variables "scsi{Min,Max}Timeout". These specify (in microseconds) the global minimum and maximum timeout periods, which override (clip) the value specified for a transaction. They may be changed at any time and affect all transactions issued after the new values are set. The range of both these variable is 0 to 0xffffffff (zero to about 4295 seconds).

SCSI TRANSACTION PRIORITY

Each transaction also has an associated priority used by the SCSI library when selecting the next command to issue when the SCSI system is idle. It chooses the highest priority transaction that can be dispatched on an available physical device. If there are several equal-priority transactions available, the SCSI library uses a simple round-robin scheme to avoid favoring the same physical device.

Priorities range from 0 (highest) to 255 (lowest), which is the same as task priorities. The priority **SCSI_THREAD_TASK_PRIORITY** can be used to give the transaction the same priority as the calling task (this is the method used internally by this SCSI-2 library).

SUPPORTED SCSI DEVICES

This library requires peripherals that conform to the SCSI-2 ANSI standard; in particular, the **INQUIRY**, **REQUEST SENSE**, and **TEST UNIT READY** commands must be supported as specified by this standard. In general, the SCSI library is self-configuring to work with any device that meets these requirements.

Peripherals that support identification and the SCSI message protocol are strongly recommended as these provide maximum performance.

In theory, all classes of SCSI devices are supported. **scsiLib** provides the capability to transact any SCSI command on any SCSI device through the **FIOSCSICOMMAND** function of the *scsiIoctl()* routine (which invokes the *scsi2Ioctl()* routine in **scsi2Lib**).

Only direct-access devices (disks) are supported by file systems like dosFs, rt11Fs and rawFs. These file systems employ routines in **scsiDirectLib** (most of which are described in **scsiLib** but defined in **scsiDirectLib**). In the case of sequential-access devices (tapes), higher-level tape file systems, like tapeFs, make use of **scsiSeqLib**. For other types of devices, additional, higher-level software is necessary to map user-level commands to SCSI transactions.

DISCONNECT/RECONNECT SUPPORT The target device can be disconnected from the SCSI bus while it carries out a SCSI command; in this way, commands to multiple SCSI devices can be overlapped to improve overall SCSI throughput. There are no restrictions on the number of pending, disconnected commands or the order in which they are resumed. The SCSI library serializes access to the device according to the capabilities and status of the device (see the following section).

Use of the disconnect/reconnect mechanism is invisible to users of the SCSI library. It can be enabled and disabled separately for each target device (see **scsiTargetOptionsSet()**). Note that support for disconnect/reconnect depends on the capabilities of the controller and its driver (see below).

TAGGED COMMAND QUEUEING SUPPORT

If the target device conforms to the ANSI SCSI-2 standard and indicates (using the INQUIRY command) that it supports command queuing, the SCSI library allows new commands to be started on the device whenever the SCSI bus is idle. That is, it executes multiple commands concurrently on the target device. By default, commands are tagged with a SIMPLE QUEUE TAG message. Up to 256 commands can be executing concurrently.

The SCSI library correctly handles contingent allegiance conditions that arise while a device is executing tagged commands. (A contingent allegiance condition exists when a target device is maintaining sense data that the initiator should use to correctly recover from an error condition.) It issues an untagged REQUEST SENSE command, and stops issuing tagged commands until the sense recovery command has completed.

For devices that do not support command queuing, the SCSI library only issues a new command when the previous one has completed. These devices can only execute a single command at once.

Use of tagged command queuing is normally invisible to users of the SCSI library. If necessary, the default tag type and maximum number of tags may be changed on a per-target basis, using **scsiTargetOptionsSet()**.

SYNCHRONOUS TRANSFER PROTOCOL SUPPORT

If the SCSI controller hardware supports the synchronous transfer protocol, **scsiLib** negotiates with the target device to determine whether to use synchronous or asynchronous transfers. Either VxWorks or the target device may start a round of negotiation. Depending on the controller hardware, synchronous transfer rates up to the maximum allowed by the SCSI-2 standard (10 Mtransfers/second) can be used.

Again, this is normally invisible to users of the SCSI library, but synchronous transfer parameters may be set or disabled on a per-target basis by using *scsiTargetOptionsSet()*.

WIDE DATA TRANSFER SUPPORT

If the SCSI controller supports the wide data transfer protocol, *scsiLib* negotiates wide data transfer parameters with the target device, if that device also supports wide transfers. Either VxWorks or the target device may start a round of negotiation. Wide data transfer parameters are negotiated prior to the synchronous data transfer parameters, as specified by the SCSI-2 ANSI specification. In conjunction with synchronous transfer, up to a maximum of 20MB/sec. can be attained.

Wide data transfer negotiation is invisible to users of this library, but it is possible to enable or disable wide data transfers and the parameters on a per-target basis by using *scsiTargetOptionsSet()*.

SCSI BUS RESET The SCSI library implements the ANSI "hard reset" option. Any transactions in progress when a SCSI bus reset is detected fail with an error code indicating termination due to bus reset. Any transactions waiting to start executing are then started normally.

CONFIGURING SCSI CONTROLLERS

The routines to create and initialize a specific SCSI controller are particular to the controller and normally are found in its library module. The normal calling sequence is:

```
xxCtrlCreate (...); /* parameters are controller specific */
xxCtrlInit (...); /* parameters are controller specific */
```

The conceptual difference between the two routines is that *xxCtrlCreate()* alloc's memory for the *xx SCSI_CTRL* data structure and initializes information that is never expected to change (for example, clock rate). The remaining fields in the *xx SCSI_CTRL* structure are initialized by *xxCtrlInit()* and any necessary registers are written on the SCSI controller to effect the desired initialization. This routine can be called multiple times, although this is rarely required. For example, the bus ID of the SCSI controller can be changed without rebooting the VxWorks system.

CONFIGURING PHYSICAL SCSI DEVICES

Before a device can be used, it must be "created," that is, declared. This is done with *scsiPhysDevCreate()* and can only be done after a *SCSI_CTRL* structure exists and has been properly initialized.

```
SCSI_PHYS_DEV *scsiPhysDevCreate
(
    SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
    int devBusId,         /* device's SCSI bus ID */
    int devLUN,           /* device's logical unit number */
    int reqSenseLength,   /* length of REQUEST SENSE data dev returns */
    int devType,          /* type of SCSI device */
    BOOL removable,      /* whether medium is removable */

```

```

    int numBlocks,      /* number of blocks on device */
    int blockSize      /* size of a block in bytes */
)

```

Several of these parameters can be left unspecified, as follows:

reqSenseLength

If 0, issue a **REQUEST_SENSE** to determine a request sense length.

devType

This parameter is ignored: an **INQUIRY** command is used to ascertain the device type. A value of **NONE** (-1) is the recommended placeholder.

numBlocks, blockSize

If 0, issue a **READ_CAPACITY** to determine the number of blocks.

The above values are recommended, unless the device does not support the required commands, or other non-standard conditions prevail.

LOGICAL PARTITIONS ON DIRECT-ACCESS BLOCK DEVICES

It is possible to have more than one logical partition on a SCSI block device. This capability is currently not supported for removable media devices. A partition is an array of contiguously addressed blocks with a specified starting block address and specified number of blocks. The *scsiBlkDevCreate()* routine is called once for each block device partition. Under normal usage, logical partitions should not overlap.

```

SCSI_BLK_DEV *scsiBlkDevCreate
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device info */
    int             numBlocks,    /* number of blocks in block device */
    int             blockOffset  /* address of first block in volume */
)

```

Note that if *numBlocks* is 0, the rest of the device is used.

ATTACHING DISK FILE SYSTEMS TO LOGICAL PARTITIONS

Files cannot be read or written to a disk partition until a file system (for example, **dosFs**, **rt11Fs**, or **rawFs**) has been initialized on the partition. For more information, see the relevant documentation in **dosFsLib**, **rt11FsLib**, or **rawFsLib**.

USING A SEQUENTIAL-ACCESS BLOCK DEVICE

The entire volume (tape) on a sequential-access block device is treated as a single raw file. This raw file is made available to higher-level layers like **tapeFs** by the *scsiSeqDevCreate()* routine, described in **scsiSeqLib**. The *scsiSeqDevCreate()* routine is called once for a given SCSI physical device.

```

SEQ_DEV *scsiSeqDevCreate
(
    SCSI_PHYS_DEV *pScsiPhysDev /* ptr to SCSI physical device info */
)

```


TRANSMITTING ARBITRARY COMMANDS TO SCSI DEVICES

The `scsi2Lib`, `scsiCommonLib`, `scsiDirectLib`, and `scsiSeqLib` libraries collectively provide routines that implement all mandatory SCSI-2 direct-access and sequential-access commands. Still, there are situations that require commands that are not supported by these libraries (for example, writing software that needs to use an optional SCSI-2 command). Arbitrary commands are handled with the `FIOSCSICOMMAND` option to `scsiIoctl()`. The `arg` parameter for `FIOSCSICOMMAND` is a pointer to a valid `SCSI_TRANSACTION` structure. Typically, a call to `scsiIoctl()` is written as a subroutine of the form:

```
STATUS myScsiCommand
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    char * buffer, /* ptr to data buffer */
    int bufLength, /* length of buffer in bytes */
    int someParam /* param. specifiable in cmd block */
)
{
    SCSI_COMMAND myScsiCmdBlock; /* SCSI command byte array */
    SCSI_TRANSACTION myScsiXaction; /* info on a SCSI transaction */
    /* fill in fields of SCSI_COMMAND structure */
    myScsiCmdBlock [0] = MY_COMMAND_OPCODE; /* the required opcode */
    .
    myScsiCmdBlock [X] = (UINT8) someParam; /* for example */
    .
    myScsiCmdBlock [N-1] = MY_CONTROL_BYTE; /* typically == 0 */
    /* fill in fields of SCSI_TRANSACTION structure */
    myScsiXaction.cmdAddress = myScsiCmdBlock;
    myScsiXaction.cmdLength = <# of valid bytes in myScsiCmdBlock>;
    myScsiXaction.dataAddress = (UINT8 *) buffer;
    myScsiXaction.dataDirection = <O_RDONLY (0) or O_WRONLY (1)>;
    myScsiXaction.dataLength = bufLength;
    myScsiXaction.addLengthByte = 0; /* no longer used */
    myScsiXaction.cmdTimeout = <timeout in usec>;
    myScsiXaction.tagType = SCSI_TAG_{DEFAULT, UNTAGGED,
                                     SIMPLE, ORDERED, HEAD_OF_Q};
    myScsiXaction.priority = [ 0 (highest) to 255 (lowest) ];
    if (scsiIoctl (pScsiPhysDev, FIOSCSICOMMAND, &myScsiXaction) == OK)
        return (OK);
    else
        /* optionally perform retry or other action based on value of
         * myScsiXaction.statusByte
         */
        return (ERROR);
}
```

INCLUDE FILES **scsiLib.h, scsi2Lib.h**

SEE ALSO **dosFsLib, rt11FsLib, rawFsLib, tapeFsLib, scsiLib, scsiCommonLib, scsiDirectLib, scsiSeqLib, scsiMgrLib, scsiCtrlLib,** *American National Standard for Information Systems – Small Computer System Interface (SCSI-2), ANSI X3T9, VxWorks Programmer’s Guide: I/O System, Local File Systems*

scsiCommonLib

NAME **scsiCommonLib** – SCSI library common commands for all devices (SCSI-2)

ROUTINES No Callable Routines

DESCRIPTION This library contains commands common to all SCSI devices. The content of this library is separated from the other SCSI libraries in order to create an additional layer for better support of all SCSI devices.

Commands in this library include:

| Command | Op Code |
|--------------------|----------------|
| INQUIRY | (0x12) |
| REQUEST | (0x03) |
| SENSE | |
| TEST UNIT READY | (0x00) |

INCLUDE FILES **scsiLib.h, scsi2Lib.h**

SEE ALSO **dosFsLib, rt11FsLib, rawFsLib, tapeFsLib, scsi2Lib,** *VxWorks Programmer’s Guide: I/O System, Local File Systems*

scsiCtrlLib

NAME **scsiCtrlLib** – SCSI thread-level controller library (SCSI-2)

ROUTINES No Callable Routines

DESCRIPTION The purpose of the SCSI controller library is to support basic SCSI controller drivers that rely on a higher level of software in order to manage SCSI transactions. More advanced

SCSI I/O processors do not require this protocol engine since software support for SCSI transactions is provided at the SCSI I/O processor level.

This library provides all the high-level routines that manage the state of the SCSI threads and guide the SCSI I/O transaction through its various stages:

- selecting a SCSI peripheral device;
- sending the identify message in order to establish the ITL nexus;
- cycling through information transfer, message and data, and status phases;
- handling bus-initiated reselects.

The various stages of the SCSI I/O transaction are reported to the SCSI manager as SCSI events. Event selection and management is handled by routines in this library.

INCLUDE FILES **scsiLib.h, scsi2Lib.h**

SEE ALSO **scsiLib, scsi2Lib, scsiCommonLib, scsiDirectLib, scsiSeqLib, scsiMgrLib**, *American National Standard for Information Systems – Small Computer System Interface (SCSI-2)*, ANSI X3T9, *VxWorks Programmer’s Guide: I/O System, Local File Systems*

scsiDirectLib

NAME **scsiDirectLib** – SCSI library for direct access devices (SCSI-2)

ROUTINES *scsiStartStopUnit()* – issue a **START_STOP_UNIT** command to a SCSI device
scsiReserve() – issue a **RESERVE** command to a SCSI device
scsiRelease() – issue a **RELEASE** command to a SCSI device

DESCRIPTION This library contains commands common to all direct-access SCSI devices. These routines are separated from **scsi2Lib** in order to create an additional layer for better support of all SCSI direct-access devices.

Commands in this library include:

| Command | Op Code |
|-----------------|----------------|
| FORMAT UNIT | (0x04) |
| READ (6) | (0x08) |
| READ (10) | (0x28) |
| READ CAPACITY | (0x25) |
| RELEASE | (0x17) |
| RESERVE | (0x16) |
| MODE SELECT (6) | (0x15) |

scsiLib

| Command | Op Code |
|------------------|----------------|
| MODE SELECT (10) | (0x55) |
| MODE SENSE (6) | (0x1a) |
| MODE SENSE (10) | (0x5a) |
| START STOP UNIT | (0x1b) |
| WRITE (6) | (0x0a) |
| WRITE (10) | (0x2a) |

INCLUDE FILES **scsiLib.h, scsi2Lib.h**

SEE ALSO **dosFsLib, rt11FsLib, rawFsLib, scsi2Lib,** *VxWorks Programmer's Guide: I/O System, Local File Systems*

scsiLib

NAME **scsiLib** – Small Computer System Interface (SCSI) library

ROUTINES

scsiPhysDevDelete() – delete a SCSI physical-device structure
scsiPhysDevCreate() – create a SCSI physical device structure
scsiPhysDevIdGet() – return a pointer to a **SCSI_PHYS_DEV** structure
scsiAutoConfig() – configure all devices connected to a SCSI controller
scsiShow() – list the physical devices attached to a SCSI controller
scsiBlkDevCreate() – define a logical partition on a SCSI block device
scsiBlkDevInit() – initialize fields in a SCSI logical partition
scsiBlkDevShow() – show the **BLK_DEV** structures on a specified physical device
scsiBusReset() – pulse the reset signal on the SCSI bus
scsiIoctl() – perform a device-specific I/O control function
scsiFormatUnit() – issue a **FORMAT_UNIT** command to a SCSI device
scsiModeSelect() – issue a **MODE_SELECT** command to a SCSI device
scsiModeSense() – issue a **MODE_SENSE** command to a SCSI device
scsiReadCapacity() – issue a **READ_CAPACITY** command to a SCSI device
scsiRdSecs() – read sector(s) from a SCSI block device
scsiWrtSecs() – write sector(s) to a SCSI block device
scsiTestUnitRdy() – issue a **TEST_UNIT_READY** command to a SCSI device
scsiInquiry() – issue an **INQUIRY** command to a SCSI device
scsiReqSense() – issue a **REQUEST_SENSE** command to a SCSI device and read results

DESCRIPTION The purpose of this library is to switch SCSI function calls (the common SCSI-1 and SCSI-2 calls listed above) to either **scsi1Lib** or **scsi2Lib**, depending upon the SCSI configuration in the Board Support Package (BSP). The normal usage is to configure SCSI-2. However,

SCSI-1 is configured when device incompatibilities exist. VxWorks can be configured with either SCSI-1 or SCSI-2, but not both SCSI-1 and SCSI-2 simultaneously.

For more information about SCSI-1 functionality, refer to **scsi1Lib**. For more information about SCSI-2, refer to **scsi2Lib**.

INCLUDE FILES **scsiLib.h, scsi1Lib.h, scsi2Lib.h**

SEE ALSO **dosFsLib, rt11FsLib, rawFsLib, scsi1Lib, scsi2Lib**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

scsiMgrLib

NAME **scsiMgrLib** – SCSI manager library (SCSI-2)

ROUTINES *scsiMgrEventNotify()* – notify the SCSI manager of a SCSI (controller) event
scsiMgrBusReset() – handle a controller-bus reset event
scsiMgrCtrlEvent() – send an event to the SCSI controller state machine
scsiMgrThreadEvent() – send an event to the thread state machine
scsiMgrShow() – show status information for the SCSI manager

DESCRIPTION This SCSI-2 library implements the SCSI manager. The purpose of the SCSI manager is to manage SCSI threads between requesting VxWorks tasks and the SCSI controller. The SCSI manager handles SCSI events and SCSI threads but allocation and de-allocation of SCSI threads is not the manager's responsibility. SCSI thread management includes despatching threads and scheduling multiple threads (which are performed by the SCSI manager, plus allocation and de-allocation of threads (which are performed by routines in **scsi2Lib**).

The SCSI manager is spawned as a VxWorks task upon initialization of the SCSI interface within VxWorks. The entry point of the SCSI manager task is *scsiMgr()*. The SCSI manager task is usually spawned during initialization of the SCSI controller driver. The driver's *xxxCtrlCreateScsi2()* routine is typically responsible for such SCSI interface initializations.

Once the SCSI manager has been initialized, it is ready to handle SCSI requests from VxWorks tasks. The SCSI manager has the following responsibilities:

- It processes requests from client tasks.
- It activates a SCSI transaction thread by appending it to the target device's wait queue and allocating a specified time period to execute a transaction.
- It handles timeout events which cause threads to be aborted.
- It receives event notifications from the SCSI driver interrupt service routine (ISR) and

scsiSeqLib

processes the event.

- It responds to events generated by the controller hardware, such as disconnection and information transfer requests.
- It replies to clients when their requests have completed or aborted.

One SCSI manager task must be spawned per SCSI controller. Thus, if a particular hardware platform contains more than one SCSI controller then that number of SCSI manager tasks must be spawned by the controller-driver initialization routine.

INCLUDE FILES **scsiLib.h, scsi2Lib.h**

SEE ALSO **scsiLib, scsi2Lib, scsiCommonLib, scsiDirectLib, scsiSeqLib, scsiCtrlLib**, *American National Standard for Information Systems – Small Computer System Interface (SCSI-2)*, ANSI X3T9, *VxWorks Programmer's Guide: I/O System, Local File Systems*

scsiSeqLib

NAME **scsiSeqLib** – SCSI sequential access device library (SCSI-2)

ROUTINES

- scsiSeqDevCreate()* – create a SCSI sequential device
- scsiErase()* – issue an ERASE command to a SCSI device
- scsiTapeModeSelect()* – issue a **MODE_SELECT** command to a SCSI tape device
- scsiTapeModeSense()* – issue a **MODE_SENSE** command to a SCSI tape device
- scsiSeqReadBlockLimits()* – issue a **READ_BLOCK_LIMITS** command to a SCSI device
- scsiRdTape()* – read bytes or blocks from a SCSI tape device
- scsiWrtTape()* – write data to a SCSI tape device
- scsiRewind()* – issue a **REWIND** command to a SCSI device
- scsiReserveUnit()* – issue a **RESERVE UNIT** command to a SCSI device
- scsiReleaseUnit()* – issue a **RELEASE UNIT** command to a SCSI device
- scsiLoadUnit()* – issue a **LOAD/UNLOAD** command to a SCSI device
- scsiWrtFileMarks()* – write file marks to a SCSI sequential device
- scsiSpace()* – move the tape on a specified physical SCSI device
- scsiSeqStatusCheck()* – detect a change in media
- scsiSeqIoctl()* – perform an I/O control function for sequential access devices

DESCRIPTION This library contains commands common to all sequential-access SCSI devices. Sequential-access SCSI devices are usually SCSI tape devices. These routines are separated from **scsi2Lib** in order to create an additional layer for better support of all SCSI sequential devices. SCSI commands in this library include:

| Command | Op Code |
|---------|---------|
| ERASE | (0x19) |

| Command | Op Code |
|------------------|---------|
| MODE SELECT (6) | (0x15) |
| MODE_SENSE (6) | (0x1a) |
| READ (6) | (0x08) |
| READ BLOCKLIMITS | (0x05) |
| RELEASE UNIT | (0x17) |
| RESERVE UNIT | (0x16) |
| REWIND | (0x01) |
| SPACE | (0x11) |
| WRITE (6) | (0x0a) |
| WRITE FILEMARKS | (0x10) |
| LOAD/UNLOAD | (0x1b) |

The SCSI routines implemented here operate mostly on a `SCSI_SEQ_DEV` structure. This structure acts as an interface between this library and a higher-level layer. The `SEQ_DEV` structure is analogous to the `BLK_DEV` structure for block devices.

The `scsiSeqDevCreate()` routine creates a `SCSI_SEQ_DEV` structure whose first element is a `SEQ_DEV`, operated upon by higher layers. This routine publishes all functions to be invoked by higher layers and maintains some state information (for example, block size) for tracking SCSI-sequential-device information.

INCLUDE FILES `scsiLib.h, scsi2Lib.h`

SEE ALSO `tapeFsLib, scsi2Lib, VxWorks Programmer's Guide: I/O System, Local File Systems`

selectLib

NAME `selectLib` – UNIX BSD 4.3 select library

ROUTINES

- `selectInit()` – initialize the select facility
- `select()` – pend on a set of file descriptors
- `selWakeup()` – wake up a task pending in `select()`
- `selWakeupAll()` – wake up all tasks in a `select()` wake-up list
- `selNodeAdd()` – add a wake-up node to a `select()` wake-up list
- `selNodeDelete()` – find and delete a node from a `select()` wake-up list
- `selWakeupListInit()` – initialize a `select()` wake-up list
- `selWakeupListLen()` – get the number of nodes in a `select()` wake-up list
- `selWakeupType()` – get the type of a `select()` wake-up node

DESCRIPTION This library provides a BSD 4.3 compatible `select` facility to wait for activity on a set of file descriptors. `selectLib` provides a mechanism that gives a driver the ability to detect

pending tasks that are awaiting activity on the driver's device. This allows a driver's interrupt service routine to wake up such tasks directly, eliminating the need for polling. The maximum number of file descriptors supported is 256.

Applications can use *select()* with pipes and serial devices, in addition to sockets. Also, *select()* examines *write* file descriptors in addition to *read* file descriptors; however, exception file descriptors remain unsupported.

Typically, application developers need concern themselves only with the *select()* call. However, driver developers should become familiar with the other routines that may be used with *select()*, if they wish to support the *select()* mechanism.

INCLUDE FILES **selectLib.h**

SEE ALSO *VxWorks Programmer's Guide: I/O System*

semBLib

NAME **semBLib** – binary semaphore library

ROUTINES *semBCreate()* – create and initialize a binary semaphore

DESCRIPTION This library provides the interface to VxWorks binary semaphores. Binary semaphores are the most versatile, efficient, and conceptually simple type of semaphore. They can be used to: (1) control mutually exclusive access to shared devices or data structures, or (2) synchronize multiple tasks, or task-level and interrupt-level processes. Binary semaphores form the foundation of numerous VxWorks facilities.

A binary semaphore can be viewed as a cell in memory whose contents are in one of two states, full or empty. When a task takes a binary semaphore, using *semTake()*, subsequent action depends on the state of the semaphore:

- (1) If the semaphore is full, the semaphore is made empty, and the calling task continues executing.
- (2) If the semaphore is empty, the task will be blocked, pending the availability of the semaphore. If a timeout is specified and the timeout expires, the pending task will be removed from the queue of pending tasks and enter the ready state with an ERROR status. A pending task is ineligible for CPU allocation. Any number of tasks may be pending simultaneously on the same binary semaphore.

When a task gives a binary semaphore, using *semGive()*, the next available task in the pending queue is unblocked. If no task is pending on this semaphore, the semaphore becomes full. Note: if a semaphore is given that unblocks a task that is of higher priority than the task that called *semGive()*, the unblocked task will preempt the calling task.

MUTUAL EXCLUSION

To use a binary semaphore as a means of mutual exclusion, first create it with an initial state of full. For example:

```
SEM_ID semMutex;  
/* create a binary semaphore that is initially full */  
semMutex = semBCreate (SEM_Q_PRIORITY, SEM_FULL);
```

Then guard a critical section or resource by taking the semaphore with *semTake()*, and exit the section or release the resource by giving the semaphore with *semGive()*. For example:

```
semTake (semMutex, WAIT_FOREVER);  
... /* critical region, accessible only by one task at a time */  
  
semGive (semMutex);
```

While there is no restriction on the same semaphore being given, taken, or flushed by multiple tasks, it is important to ensure the proper functionality of the mutual-exclusion construct. While there is no danger in any number of processes taking a semaphore, the giving of a semaphore should be more carefully controlled. If a semaphore is given by a task that did not take it, mutual exclusion could be lost.

SYNCHRONIZATION To use a binary semaphore as a means of synchronization, create it with an initial state of empty. A task blocks by taking a semaphore at a synchronization point, and it remains blocked until the semaphore is given by another task or interrupt service routine.

Synchronization with interrupt service routines is a particularly common need. Binary semaphores can be given, but not taken, from interrupt level. Thus, a task can block at a synchronization point with *semTake()*, and an interrupt service routine can unblock that task with *semGive()*.

In the following example, when *init()* is called, the binary semaphore is created, an interrupt service routine is attached to an event, and a task is spawned to process the event. Task 1 will run until it calls *semTake()*, at which point it will block until an event causes the interrupt service routine to call *semGive()*. When the interrupt service routine completes, task 1 can execute to process the event.

```
SEM_ID semSync;    /* ID of sync semaphore */  
init ()  
{  
    intConnect (... , eventInterruptSvcRout, ...);  
    semSync = semBCreate (SEM_Q_FIFO, SEM_EMPTY);  
    taskSpawn (... , task1);  
}  
task1 ()  
{  
    ...  
    semTake (semSync, WAIT_FOREVER);    /* wait for event */
```

```
        ...    /* process event */  
    }  
    eventInterruptSvcRout ()  
    {  
        ...  
        semGive (semSync);    /* let task 1 process event */  
        ...  
    }
```

A *semFlush()* on a binary semaphore will atomically unblock all pended tasks in the semaphore queue, i.e., all tasks will be unblocked at once, before any actually execute.

CAVEATS

There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible. Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus, if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores will not be given back, effectively leaving a resource permanently unavailable. The mutual-exclusion semaphores provided by **semMLib** offer protection from unexpected task deletion.

INCLUDE FILES

semLib.h

SEE ALSO

semLib, **semCLib**, **semMLib**, *VxWorks Programmer's Guide: Basic OS*

semCLib

NAME

semCLib – counting semaphore library

ROUTINES

semCCreate() – create and initialize a counting semaphore

DESCRIPTION

This library provides the interface to VxWorks counting semaphores. Counting semaphores are useful for guarding multiple instances of a resource.

A counting semaphore may be viewed as a cell in memory whose contents keep track of a count. When a task takes a counting semaphore, using *semTake()*, subsequent action depends on the state of the count:

- (1) If the count is non-zero, it is decremented and the calling task continues executing.
- (2) If the count is zero, the task will be blocked, pending the availability of the semaphore. If a timeout is specified and the timeout expires, the pended task will be removed from the queue of pended tasks and enter the ready state with an ERROR status. A pended task is ineligible for CPU allocation. Any number of tasks may be pended simultaneously on the same counting semaphore.

When a task gives a semaphore, using *semGive()*, the next available task in the pend queue is unblocked. If no task is pending on this semaphore, the semaphore count is incremented. Note that if a semaphore is given, and a task is unblocked that is of higher priority than the task that called *semGive()*, the unblocked task will preempt the calling task.

A *semFlush()* on a counting semaphore will atomically unblock all pended tasks in the semaphore queue. So all tasks will be made ready before any task actually executes. The count of the semaphore will remain unchanged.

INTERRUPT USAGE Counting semaphores may be given but not taken from interrupt level.

CAVEATS There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible. Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus, if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores will not be given back, effectively leaving a resource permanently unavailable. The mutual-exclusion semaphores provided by **semMLib** offer protection from unexpected task deletion.

INCLUDE FILES **semLib.h**

SEE ALSO **semLib**, **semBLib**, **semMLib**, *VxWorks Programmer's Guide: Basic OS*

semLib

NAME **semLib** – general semaphore library

ROUTINES *semGive()* – give a semaphore
semTake() – take a semaphore
semFlush() – unblock every task pended on a semaphore
semDelete() – delete a semaphore

DESCRIPTION Semaphores are the basis for synchronization and mutual exclusion in VxWorks. They are powerful in their simplicity and form the foundation for numerous VxWorks facilities.

Different semaphore types serve different needs, and while the behavior of the types differs, their basic interface is the same. This library provides semaphore routines common to all VxWorks semaphore types. For all types, the two basic operations are *semTake()* and *semGive()*, the acquisition or relinquishing of a semaphore.

semLib

Semaphore creation and initialization is handled by other libraries, depending on the type of semaphore used. These libraries contain full functional descriptions of the semaphore types:

- semBLib** – binary semaphores
- semCLib** – counting semaphores
- semMLib** – mutual exclusion semaphores
- semSmLib** – shared memory semaphores

Binary semaphores offer the greatest speed and the broadest applicability.

The **semLib** library provides all other semaphore operations, including routines for semaphore control, deletion, and information. Semaphores must be validated before any semaphore operation can be undertaken. An invalid semaphore ID results in **ERROR**, and an appropriate **errno** is set.

SEMAPHORE CONTROL

The *semTake()* call acquires a specified semaphore, blocking the calling task or making the semaphore unavailable. All semaphore types support a timeout on the *semTake()* operation. The timeout is specified as the number of ticks to remain blocked on the semaphore. Timeouts of **WAIT_FOREVER** and **NO_WAIT** codify common timeouts. If a *semTake()* times out, it returns **ERROR**. Refer to the library of the specific semaphore type for the exact behavior of this operation.

The *semGive()* call relinquishes a specified semaphore, unblocking a pended task or making the semaphore available. Refer to the library of the specific semaphore type for the exact behavior of this operation.

The *semFlush()* call may be used to atomically unblock all tasks pended on a semaphore queue, i.e., all tasks will be unblocked before any are allowed to run. It may be thought of as a broadcast operation in synchronization applications. The state of the semaphore is unchanged by the use of *semFlush()*; it is not analogous to *semGive()*.

SEMAPHORE DELETION

The *semDelete()* call terminates a semaphore and deallocates any associated memory. The deletion of a semaphore unblocks tasks pended on that semaphore; the routines which were pended return **ERROR**. Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that already has taken (owns) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully taken.

SEMAPHORE INFORMATION

The *semInfo()* call is a useful debugging aid, reporting all tasks blocked on a specified semaphore. It provides a snapshot of the queue at the time of the call, but because semaphores are dynamic, the information may be out of date by the time it is available. As with the current state of the semaphore, use of the queue of pended tasks should be restricted to debugging uses only.

INCLUDE FILES `semLib.h`

SEE ALSO `taskLib`, `semBLib`, `semCLib`, `semMLib`, `semSmLib`, *VxWorks Programmer's Guide:Basic OS*

semMLib

NAME `semMLib` – mutual-exclusion semaphore library

ROUTINES `semMCreate()` – create and initialize a mutual-exclusion semaphore
`semMGiveForce()` – give a mutual-exclusion semaphore without restrictions

DESCRIPTION This library provides the interface to VxWorks mutual-exclusion semaphores. Mutual-exclusion semaphores offer convenient options suited for situations requiring mutually exclusive access to resources. Typical applications include sharing devices and protecting data structures. Mutual-exclusion semaphores are used by many higher-level VxWorks facilities.

The mutual-exclusion semaphore is a specialized version of the binary semaphore, designed to address issues inherent in mutual exclusion, such as recursive access to resources, priority inversion, and deletion safety. The fundamental behavior of the mutual-exclusion semaphore is identical to the binary semaphore (see the manual entry for `semBLib`), except for the following restrictions:

- It can only be used for mutual exclusion.
- It can only be given by the task that took it.
- It may not be taken or given from interrupt level.
- The `semFlush()` operation is illegal.

These last two operations have no meaning in mutual-exclusion situations.

RECURSIVE RESOURCE ACCESS

A special feature of the mutual-exclusion semaphore is that it may be taken "recursively," i.e., it can be taken more than once by the task that owns it before finally being released. Recursion is useful for a set of routines that need mutually exclusive access to a resource, but may need to call each other.

Recursion is possible because the system keeps track of which task currently owns a mutual-exclusion semaphore. Before being released, a mutual-exclusion semaphore taken recursively must be given the same number of times it has been taken; this is tracked by means of a count which is incremented with each `semTake()` and decremented with each `semGive()`.

The example below illustrates recursive use of a mutual-exclusion semaphore. Function A requires access to a resource which it acquires by taking `semM`; function A may also need to call function B, which also requires `semM`:

```
SEM_ID semM;  
semM = semMCreate (...);  
funcA ()  
{  
    semTake (semM, WAIT_FOREVER);  
    ...  
    funcB ();  
    ...  
    semGive (semM);  
}  
funcB ()  
{  
    semTake (semM, WAIT_FOREVER);  
    ...  
    semGive (semM);  
}
```

PRIORITY-INVERSION SAFETY

If the option `SEM_INVERSION_SAFE` is selected, the library adopts a priority-inheritance protocol to resolve potential occurrences of "priority inversion," a problem stemming from the use of semaphores for mutual exclusion. Priority inversion arises when a higher-priority task is forced to wait an indefinite period of time for the completion of a lower-priority task.

Consider the following scenario: T1, T2, and T3 are tasks of high, medium, and low priority, respectively. T3 has acquired some resource by taking its associated semaphore. When T1 preempts T3 and contends for the resource by taking the same semaphore, it becomes blocked. If we could be assured that T1 would be blocked no longer than the time it normally takes T3 to finish with the resource, the situation would not be problematic. However, the low-priority task is vulnerable to preemption by medium-priority tasks; a preempting task, T2, could inhibit T3 from relinquishing the resource. This condition could persist, blocking T1 for an indefinite period of time.

The priority-inheritance protocol solves the problem of priority inversion by elevating the priority of T3 to the priority of T1 during the time T1 is blocked on T3. This protects T3, and indirectly T1, from preemption by T2. Stated more generally, the priority-inheritance protocol assures that a task which owns a resource will execute at the priority of the highest priority task blocked on that resource. Once the task priority has been elevated, it remains at the higher level until all mutual-exclusion semaphores that the task owns are released; then the task returns to its normal, or standard, priority. Hence, the "inheriting" task is protected from preemption by any intermediate-priority tasks.

The priority-inheritance protocol also takes into consideration a task's ownership of more than one mutual-exclusion semaphore at a time. Such a task will execute at the priority of the highest priority task blocked on any of its owned resources. The task will return to its normal priority only after relinquishing all of its mutual-exclusion semaphores that have the inversion-safety option enabled.

SEMAPHORE DELETION

The *semDelete()* call terminates a semaphore and deallocates any associated memory. The deletion of a semaphore unblocks tasks pending on that semaphore; the routines which were pending return ERROR. Take special care when deleting mutual-exclusion semaphores to avoid deleting a semaphore out from under a task that already owns (has taken) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task owns.

TASK-DELETION SAFETY

If the option **SEM_DELETE_SAFE** is selected, the task owning the semaphore will be protected from deletion as long as it owns the semaphore. This solves another problem endemic to mutual exclusion. Deleting a task executing in a critical region can be catastrophic. The resource could be left in a corrupted state and the semaphore guarding the resource would be unavailable, effectively shutting off all access to the resource.

As discussed in *taskLib*, the primitives *taskSafe()* and *taskUnsafe()* offer one solution, but as this type of protection goes hand in hand with mutual exclusion, the mutual-exclusion semaphore provides the option **SEM_DELETE_SAFE**, which enables an implicit *taskSafe()* with each *semTake()*, and a *taskUnsafe()* with each *semGive()*. This convenience is also more efficient, as the resulting code requires fewer kernel entrances.

CAVEATS

There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible. Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores will not be given back, effectively leaving a resource permanently unavailable. The **SEM_DELETE_SAFE** option partially protects an application, to the extent that unexpected deletions will be deferred until the resource is released.

Because the priority of a task which has been elevated by the taking of a mutual-exclusion semaphore remains at the higher priority until all mutexes held by that task are released, unbounded priority inversion situations can result when nested mutexes are involved. If nested mutexes are required, consider the following alternatives:

1. Avoid overlapping critical regions.
2. Adjust priorities of tasks so that there are no tasks at intermediate priority levels.
3. Adjust priorities of tasks so that priority inheritance protocol is not needed.
4. Manually implement a static priority ceiling protocol using a non-inversion-safe mutex. This involves setting all blockers on a mutex to the ceiling priority, then taking the mutex. After *semGive*, set the priorities back to the base priority. Note that this implementation reduces the queue to a fifo queue.

INCLUDE FILES **semLib.h**

SEE ALSO **semLib, semBLib, semCLib, VxWorks Programmer's Guide: Basic OS**

semOLib

| | |
|----------------------|--|
| NAME | semOLib – release 4.x binary semaphore library |
| ROUTINES | <i>semCreate()</i> – create and initialize a release 4.x binary semaphore <i>semInit()</i> – initialize a static binary semaphore <i>semClear()</i> – take a release 4.x semaphore, if the semaphore is available |
| DESCRIPTION | <p>This library is provided for backward compatibility with VxWorks 4.x semaphores. The semaphores are identical to 5.0 binary semaphores, except that timeouts -- missing or specified -- are ignored.</p> <p>For backward compatibility, <i>semCreate()</i> operates as before, allocating and initializing a 4.x-style semaphore. Likewise, <i>semClear()</i> has been implemented as a <i>semTake()</i>, with a timeout of <code>NO_WAIT</code>.</p> <p>For more information on of the behavior of binary semaphores, see the manual entry for semBLib.</p> |
| INCLUDE FILES | semLib.h |
| SEE ALSO | semLib , semBLib , <i>VxWorks Programmer's Guide: Basic OS</i> |

semPxBLib

| | |
|--------------------|---|
| NAME | semPxBLib – semaphore synchronization library (POSIX) |
| ROUTINES | <i>semPxBLibInit()</i> – initialize POSIX semaphore support <i>sem_init()</i> – initialize an unnamed semaphore (POSIX) <i>sem_destroy()</i> – destroy an unnamed semaphore (POSIX) <i>sem_open()</i> – initialize/open a named semaphore (POSIX) <i>sem_close()</i> – close a named semaphore (POSIX) <i>sem_unlink()</i> – remove a named semaphore (POSIX) <i>sem_wait()</i> – lock (take) a semaphore, blocking if not available (POSIX) <i>sem_trywait()</i> – lock (take) a semaphore, returning error if unavailable (POSIX) <i>sem_post()</i> – unlock (give) a semaphore (POSIX) <i>sem_getvalue()</i> – get the value of a semaphore (POSIX) |
| DESCRIPTION | This library implements the POSIX 1003.1b semaphore interface. For alternative semaphore routines designed expressly for VxWorks, see the manual page for semLib |

and other semaphore libraries mentioned there. POSIX semaphores are counting semaphores; as such they are most similar to the **semCLib** VxWorks-specific semaphores.

The main advantage of POSIX semaphores is portability (to the extent that alternative operating systems also provide these POSIX interfaces). However, VxWorks-specific semaphores provide the following features absent from the semaphores implemented in this library: priority inheritance, task-deletion safety, the ability for a single task to take a semaphore multiple times, ownership of mutual-exclusion semaphores, semaphore timeout, and the choice of queuing mechanism.

POSIX defines both named and unnamed semaphores; **semPxLib** includes separate routines for creating and deleting each kind. For other operations, applications use the same routines for both kinds of semaphore.

TERMINOLOGY The POSIX standard uses the terms *wait* or *lock* where *take* is normally used in VxWorks, and the terms *post* or *unlock* where *give* is normally used in VxWorks. VxWorks documentation that is specific to the POSIX interfaces (such as the remainder of this manual entry, and the manual entries for subroutines in this library) uses the POSIX terminology, in order to make it easier to read in conjunction with other references on POSIX.

SEMAPHORE DELETION

The *sem_destroy()* call terminates an unnamed semaphore and deallocates any associated memory; the combination of *sem_close()* and *sem_unlink()* has the same effect for named semaphores. Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that has already locked that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully locked. (Similarly, for named semaphores, applications should take care to only close semaphores that the closing task has opened.)

If there are tasks blocked waiting for the semaphore, *sem_destroy()* fails and sets **errno** to **EBUSY**.

INCLUDE FILES **semaphore.h**

SEE ALSO POSIX 1003.1b document, **semLib**, *VxWorks Programmer's Guide: Basic OS*

semPxShow

| | |
|--------------------|---|
| NAME | semPxShow – POSIX semaphore show library |
| ROUTINES | <i>semPxShowInit()</i> – initialize the POSIX semaphore show facility |
| DESCRIPTION | This library provides a show routine for POSIX semaphore objects. |

semShow

| | |
|----------------------|---|
| NAME | semShow – semaphore show routines |
| ROUTINES | <i>semShowInit()</i> – initialize the semaphore show facility <i>semInfo()</i> – get a list of task IDs that are blocked on a semaphore <i>semShow()</i> – show information about a semaphore |
| DESCRIPTION | <p>This library provides routines to show semaphore statistics, such as semaphore type, semaphore queuing method, tasks pended, etc.</p> <p>The routine <i>semShowInit()</i> links the semaphore show facility into the VxWorks system. It is called automatically when the semaphore show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none">– If you use the configuration header files, define INCLUDE_SHOW_ROUTINES in config.h.– If you use the Tornado project facility, select INCLUDE_SEM_SHOW. |
| INCLUDE FILES | semLib.h |
| SEE ALSO | semLib , <i>VxWorks Programmer's Guide: Basic OS</i> |

semSmLib

| | |
|----------------------------|---|
| NAME | semSmLib – shared memory semaphore library (VxMP Opt.) |
| ROUTINES | <i>semBSmCreate()</i> – create and initialize a shared memory binary semaphore (VxMP Opt.) <i>semCSmCreate()</i> – create and initialize a shared memory counting semaphore (VxMP Opt.) |
| DESCRIPTION | <p>This library provides the interface to VxWorks shared memory binary and counting semaphores. Once a shared memory semaphore is created, the generic semaphore-handling routines provided in semLib are used to manipulate it. Shared memory binary semaphores are created using <i>semBSmCreate()</i>. Shared memory counting semaphores are created using <i>semCSmCreate()</i>.</p> <p>Shared memory binary semaphores are used to: (1) control mutually exclusive access to multiprocessor-shared data structures, or (2) synchronize multiple tasks running in a multiprocessor system. For general information about binary semaphores, see the manual entry semBLib.</p> <p>Shared memory counting semaphores are used for guarding multiple instances of a resource used by multiple CPUs. For general information about shared counting semaphores, see the manual entry for semCLib.</p> <p>For information about the generic semaphore-handling routines, see the manual entry for semLib.</p> |
| MEMORY REQUIREMENTS | <p>The semaphore structure is allocated from a dedicated shared memory partition.</p> <p>The shared semaphore dedicated shared memory partition is initialized by the shared memory objects master CPU. The size of this partition is defined by the maximum number of shared semaphores, defined by SM_OBJ_MAX_SEM in the configuration header file.</p> <p>This memory partition is common to shared binary and counting semaphores, thus SM_OBJ_MAX_SEM must be set to the sum total of binary and counting semaphores to be used in the system.</p> |
| RESTRICTIONS | <p>Shared memory semaphores differ from local semaphores in the following ways:</p> <p>Interrupt Use. Shared semaphores may not be given, taken, or flushed at interrupt level.</p> <p>Deletion. There is no way to delete a shared semaphore and free its associated shared memory. Attempts to delete a shared semaphore return ERROR and set errno to S_smObjLib_NO_OBJECT_DESTROY.</p> |

Queuing Style.

The shared semaphore queuing style specified when the semaphore is created must be FIFO.

INTERRUPT LATENCY

Internally, interrupts are locked while manipulating shared semaphore data structures, thus increasing the interrupt latency.

CONFIGURATION

Before routines in this library can be called, the shared memory object facility must be initialized by calling *usrSmObjInit()*, which is found in *src/config/usrSmObj.c*. This is done automatically from the root task, *usrRoot()*, in *usrConfig.c* when the configuration macro *INCLUDE_SM_OBJ* is defined.

AVAILABILITY

This module is distributed as a component of the unbundled shared memory support option, VxMP.

INCLUDE FILES

semSmLib.h

SEE ALSO

semLib, *semBLib*, *semCLib*, *smObjLib*, *semShow*, *usrSmObjInit()*, *VxWorks Programmer's Guide: Shared Memory Objects, Basic OS*

shellLib

NAME

shellLib – shell execution routines

ROUTINES

shellInit() – start the shell
shell() – the shell entry point
shellScriptAbort() – signal the shell to stop processing a script
shellHistory() – display or set the size of shell history
shellPromptSet() – change the shell prompt
shellOrigStdSet() – set the shell's default input/output/error file descriptors
shellLock() – lock access to the shell

DESCRIPTION

This library contains the execution support routines for the VxWorks shell. It provides the basic programmer's interface to VxWorks. It is a C-expression interpreter, containing no built-in commands.

The nature, use, and syntax of the shell are more fully described in the *VxWorks Programmer's Guide: Target Shell*.

INCLUDE FILES

shellLib.h

SEE ALSO

ledLib, *VxWorks Programmer's Guide: Target Shell*

sigLib

NAME sigLib – software signal facility library

ROUTINES

- sigInit()* – initialize the signal facilities
- sigqueueInit()* – initialize the queued signal facilities
- sigemptyset()* – initialize a signal set with no signals included (POSIX)
- sigfillset()* – initialize a signal set with all signals included (POSIX)
- sigaddset()* – add a signal to a signal set (POSIX)
- sigdelset()* – delete a signal from a signal set (POSIX)
- sigismember()* – test to see if a signal is in a signal set (POSIX)
- signal()* – specify the handler associated with a signal
- sigaction()* – examine and/or specify the action associated with a signal (POSIX)
- sigprocmask()* – examine and/or change the signal mask (POSIX)
- sigpending()* – retrieve the set of pending signals blocked from delivery (POSIX)
- sigsuspend()* – suspend the task until delivery of a signal (POSIX)
- pause()* – suspend the task until delivery of a signal (POSIX)
- sigtimedwait()* – wait for a signal
- sigwaitinfo()* – wait for real-time signals
- sigvec()* – install a signal handler
- sigsetmask()* – set the signal mask
- sigblock()* – add to a set of blocked signals
- raise()* – send a signal to the caller's task
- kill()* – send a signal to a task (POSIX)
- sigqueue()* – send a queued signal to a task

DESCRIPTION This library provides a signal interface for tasks. Signals are used to alter the flow control of tasks by communicating asynchronous events within or between task contexts. Any task or interrupt service can "raise" (or send) a signal to a particular task. The task being signaled will immediately suspend its current thread of execution and invoke a task-specified "signal handler" routine. The signal handler is a user-supplied routine that is bound to a specific signal and performs whatever actions are necessary whenever the signal is received. Signals are most appropriate for error and exception handling, rather than as a general purpose intertask communication mechanism.

This library has both a BSD 4.3 and POSIX signal interface. The POSIX interface provides a standardized interface which is more functional than the traditional BSD 4.3 interface. The chart below shows the correlation between BSD 4.3 and POSIX 1003.1 functions. An application should use only one form of interface and not intermix them.

| BSD 4.3 | POSIX 1003.1 |
|---------------------|---|
| <i>sigmask()</i> | <i>sigemptyset()</i> , <i>sigfillset()</i> , <i>sigaddset()</i> , <i>sigdelset()</i> , <i>sigismember()</i> |
| <i>sigblock()</i> | <i>sigprocmask()</i> |
| <i>sigsetmask()</i> | <i>sigprocmask()</i> |

| BSD 4.3 | POSIX 1003.1 |
|-----------------|---------------------|
| <i>pause()</i> | <i>sigsuspend()</i> |
| <i>sigvec()</i> | <i>sigaction()</i> |
| (none) | <i>sigpending()</i> |
| <i>signal()</i> | <i>signal()</i> |
| <i>kill()</i> | <i>kill()</i> |

POSIX 1003.1b (Real-Time Extensions) also specifies a queued-signal facility that involves four additional routines: *sigqueue()*, *sigwaitinfo()*, and *sigtimedwait()*.

In many ways, signals are analogous to hardware interrupts. The signal facility provides a set of 31 distinct signals. A signal can be raised by calling *kill()*, which is analogous to an interrupt or hardware exception. A signal handler is bound to a particular signal with *sigaction()* in much the same way that an interrupt service routine is connected to an interrupt vector with *intConnect()*. Signals are blocked for the duration of the signal handler, just as interrupts are locked out for the duration of the interrupt service routine. Tasks can block the occurrence of certain signals with *sigprocmask()*, just as the interrupt level can be raised or lowered to block out levels of interrupts. If a signal is blocked when it is raised, its handler routine will be called when the signal becomes unblocked.

Several routines (*sigprocmask()*, *sigpending()*, and *sigsuspend()*) take *sigset_t* data structures as parameters. These data structures are used to specify signal set masks. Several routines are provided for manipulating these data structures: *sigemptyset()* clears all the bits in a *sigset_t*, *sigfillset()* sets all the bits in a *sigset_t*, *sigaddset()* sets the bit in a *sigset_t* corresponding to a particular signal number, *sigdelset()* resets the bit in a *sigset_t* corresponding to a particular signal number, and *sigismember()* tests to see if the bit corresponding to a particular signal number is set.

FUNCTION RESTARTING

If a task is pended (for instance, by waiting for a semaphore to become available) and a signal is sent to the task for which the task has a handler installed, then the handler will run before the semaphore is taken. When the handler is done, the task will go back to being pended (waiting for the semaphore). If there was a timeout used for the pend, then the original value will be used again when the task returns from the signal handler and goes back to being pended.

Signal handlers are typically defined as:

```
void sigHandler
(
    int sig, /* signal number */
)
{
    ...
}
```

In VxWorks, the signal handler is passed additional arguments and can be defined as:

```
void sigHandler
(
    int sig,                /* signal number          */
    int code,              /* additional code       */
    struct sigcontext *pSigContext /* context of task before signal */
)
{
    ...
}
```

The parameter *code* is valid only for signals caused by hardware exceptions. In this case, it is used to distinguish signal variants. For example, both numeric overflow and zero divide raise **SIGFPE** (floating-point exception) but have different values for *code*. (Note that when the above VxWorks extensions are used, the compiler may issue warnings.)

SIGNAL HANDLER DEFINITION

Signal handling routines must follow one of two specific formats, so that they may be correctly called by the operating system when a signal occurs.

Traditional signal handlers receive the signal number as the sole input parameter. However, certain signals generated by routines which make up the POSIX Real-Time Extensions (P1003.1b) support the passing of an additional application-specific value to the handler routine. These include signals generated by the *sigqueue()* call, by asynchronous I/O, by POSIX real-time timers, and by POSIX message queues.

If a signal handler routine is to receive these additional parameters, **SA_SIGINFO** must be set in the *sa_flags* field of the *sigaction* structure which is a parameter to the *sigaction()* routine. Such routines must take the following form:

```
void sigHandler (int sigNum, siginfo_t * pInfo, void * pContext);
```

Traditional signal handling routines must not set **SA_SIGINFO** in the *sa_flags* field, and must take the form of:

```
void sigHandler (int sigNum);
```

EXCEPTION PROCESSING

Certain signals, defined below, are raised automatically when hardware exceptions are encountered. This mechanism allows user-defined exception handlers to be installed. This is useful for recovering from catastrophic events such as bus or arithmetic errors. Typically, *setjmp()* is called to define the point in the program where control will be restored, and *longjmp()* is called in the signal handler to restore that context. Note that *longjmp()* restores the state of the task's signal mask. If a user-defined handler is not installed or the installed handler returns for a signal raised by a hardware exception, then the task is suspended and a message is logged to the console.

The following is a list of hardware exceptions caught by VxWorks and delivered to the offending task. The user may include the higher-level header file **sigCodes.h** in order to access the appropriate architecture-specific header file containing the code value.

Motorola 68K

| Signal | Code | Exception |
|---------------|--------------------|--------------------------------|
| SIGSEGV | NULL | bus error |
| SIGBUS | BUS_ADDERR | address error |
| SIGILL | ILL_ILLINSTR_FAULT | illegal instruction |
| SIGFPE | FPE_INTDIV_TRAP | zero divide |
| SIGFPE | FPE_CHKINST_TRAP | chk trap |
| SIGFPE | FPE_TRAPV_TRAP | trapv trap |
| SIGILL | ILL_PRIVVIO_FAULT | privilege violation |
| SIGTRAP | NULL | trace exception |
| SIGEMT | EMT_EMU1010 | line 1010 emulator |
| SIGEMT | EMT_EMU1111 | line 1111 emulator |
| SIGILL | ILL_ILLINSTR_FAULT | coprocessor protocol violation |
| SIGFMT | NULL | format error |
| SIGFPE | FPE_FLTBSUN_TRAP | compare unordered |
| SIGFPE | FPE_FLTINEX_TRAP | inexact result |
| SIGFPE | FPE_FLTDIV_TRAP | divide by zero |
| SIGFPE | FPE_FLTUND_TRAP | underflow |
| SIGFPE | FPE_FLTOPERR_TRAP | operand error |
| SIGFPE | FPE_FLTOVF_TRAP | overflow |
| SIGFPE | FPE_FLTNAN_TRAP | signaling "Not A Number" |

SPARC

| Signal | Code | Exception |
|---------------|---------------------|--------------------------------|
| SIGBUS | BUS_INSTR_ACCESS | bus error on instruction fetch |
| SIGBUS | BUS_ALIGN | address error (bad alignment) |
| SIGBUS | BUS_DATA_ACCESS | bus error on data access |
| SIGILL | ILL_ILLINSTR_FAULT | illegal instruction |
| SIGILL | ILL_PRIVINSTR_FAULT | privilege violation |
| SIGILL | ILL_COPROC_DISABLED | coprocessor disabled |
| SIGILL | ILL_COPROC_EXCPTN | coprocessor exception |
| SIGILL | ILL_TRAP_FAULT(n) | uninitialized user trap |
| SIGFPE | FPE_FPA_ENABLE | floating point disabled |
| SIGFPE | FPE_FPA_ERROR | floating point exception |
| SIGFPE | FPE_INTDIV_TRAP | zero divide |
| SIGEMT | EMT_TAG | tag overflow |

Intel i960

| Signal | Code | Exception |
|---------------|---------------|-------------------------------|
| SIGBUS | BUS_UNALIGNED | address error (bad alignment) |
| SIGBUS | BUS_BUSERR | bus error |

| Signal | Code | Exception |
|---------|--------------------------------|----------------------------------|
| SIGILL | ILL_INVALID_OPCODE | invalid instruction |
| SIGILL | ILL_UNIMPLEMENTED | instr fetched from on-chip RAM |
| SIGILL | ILL_INVALID_OPERAND | invalid operand |
| SIGILL | ILL_CONSTRAINT_RANGE | constraint range failure |
| SIGILL | ILL_PRIVILEGED | privilege violation |
| SIGILL | ILL_LENGTH | bad index to sys procedure table |
| SIGILL | ILL_TYPE_MISMATCH | privilege violation |
| SIGTRAP | TRAP_INSTRUCTION_TRACE | instruction trace fault |
| SIGTRAP | TRAP_BRANCH_TRACE | branch trace fault |
| SIGTRAP | TRAP_CALL_TRACE | call trace fault |
| SIGTRAP | TRAP_RETURN_TRACE | return trace fault |
| SIGTRAP | TRAP_PRERETURN_TRACE | pre-return trace fault |
| SIGTRAP | TRAP_SUPERVISOR_TRACE | supervisor trace fault |
| SIGTRAP | TRAP_BREAKPOINT_TRACE | breakpoint trace fault |
| SIGFPE | FPE_INTEGER_OVERFLOW | integer overflow |
| SIGFPE | FST_ZERO_DIVIDE | integer zero divide |
| SIGFPE | FPE_FLOATING_OVERFLOW | floating point overflow |
| SIGFPE | FPE_FLOATING_UNDERFLOW | floating point underflow |
| SIGFPE | FPE_FLOATING_INVALID_OPERATION | invalid floating point operation |
| SIGFPE | FPE_FLOATING_ZERO_DIVIDE | floating point zero divide |
| SIGFPE | FPE_FLOATING_INEXACT | floating point inexact |
| SIGFPE | FPE_FLOATING_RESERVED_ENCODING | floating point reserved encoding |

MIPS R3000/R4000

| Signal | Code | Exception |
|---------|---------------------|--|
| SIGBUS | BUS_TLBMOD | TLB modified |
| SIGBUS | BUS_TLBL | TLB miss on a load instruction |
| SIGBUS | BUS_TLBS | TLB miss on a store instruction |
| SIGBUS | BUS_ADEL | address error (bad alignment) on load instr |
| SIGBUS | BUS_ADES | address error (bad alignment) on store instr |
| SIGSEGV | SEGV_IBUS | bus error (instruction) |
| SIGSEGV | SEGV_DBUS | bus error (data) |
| SIGTRAP | TRAP_SYSCALL | syscall instruction executed |
| SIGTRAP | TRAP_BP | break instruction executed |
| SIGILL | ILL_ILLINSTR_FAULT | reserved instruction |
| SIGILL | ILL_COPROC_UNUSABLE | coprocessor unusable |
| SIGFPE | FPE_FPA_UIO, SIGFPE | unimplemented FPA operation |
| SIGFPE | FPE_FLTNAN_TRAP | invalid FPA operation |
| SIGFPE | FPE_FLTDIV_TRAP | FPA divide by zero |
| SIGFPE | FPE_FLTOVF_TRAP | FPA overflow exception |

| Signal | Code | Exception |
|--------|------------------|-------------------------|
| SIGFPE | FPE_FLTUND_TRAP | FPA underflow exception |
| SIGFPE | FPE_FLTINEX_TRAP | FPA inexact operation |

Intel i386/i486

| Signal | Code | Exception |
|--------|----------------------|-----------------------------|
| SIGILL | ILL_DIVIDE_ERROR | divide error |
| SIGEMT | EMT_DEBUG | debugger call |
| SIGILL | ILL_NON_MASKABLE | NMI interrupt |
| SIGEMT | EMT_BREAKPOINT | breakpoint |
| SIGILL | ILL_OVERFLOW | INTO-detected overflow |
| SIGILL | ILL_BOUND | bound range exceeded |
| SIGILL | ILL_INVALID_OPCODE | invalid opcode |
| SIGFPE | FPE_NO_DEVICE | device not available |
| SIGILL | ILL_DOUBLE_FAULT | double fault |
| SIGFPE | FPE_CP_OVERRUN | coprocessor segment overrun |
| SIGILL | ILL_INVALID_TSS | invalid task state segment |
| SIGBUS | BUS_NO_SEGMENT | segment not present |
| SIGBUS | BUS_STACK_FAULT | stack exception |
| SIGILL | ILL_PROTECTION_FAULT | general protection |
| SIGBUS | BUS_PAGE_FAULT | page fault |
| SIGILL | ILL_RESERVED | (intel reserved) |
| SIGFPE | FPE_CP_ERROR | coprocessor error |
| SIGBUS | BUS_ALIGNMENT | alignment check |

PowerPC

| Signal | Code | Exception |
|---------|-------------------|---|
| SIGBUS | _EXC_OFF_MACH | machine check |
| SIGBUS | _EXC_OFF_INST | instruction access |
| SIGBUS | _EXC_OFF_ALIGN | alignment |
| SIGILL | _EXC_OFF_PROG | program |
| SIGBUS | _EXC_OFF_DATA | data access |
| SIGFPE | _EXC_OFF_FPU | floating point unavailable |
| SIGTRAP | _EXC_OFF_DBG | debug exception (PPC403) |
| SIGTRAP | _EXC_OFF_INST_BRK | inst. breakpoint (PPC603, PPCEC603, PPC604) |
| SIGTRAP | _EXC_OFF_TRACE | trace (PPC603, PPCEC603, PPC604, PPC860) |
| SIGBUS | _EXC_OFF_CRTL | critical interrupt (PPC403) |
| SIGILL | _EXC_OFF_SYSCALL | system call |

INCLUDE FILES `signal.h`

SEE ALSO `intLib`, IEEE *POSIX 1003.1b*, *VxWorks Programmer's Guide: Basic OS*

smMemLib

| | |
|----------------------|--|
| NAME | smMemLib – shared memory management library (VxMP Opt.) |
| ROUTINES | <i>memPartSmCreate()</i> – create a shared memory partition <i>smMemAddToPool()</i> – add memory to the shared memory system partition <i>smMemOptionsSet()</i> – set the debug options for the shared memory system partition <i>smMemMalloc()</i> – allocate a block of memory from the shared memory system partition <i>smMemCalloc()</i> – allocate memory for an array from the shared memory system partition <i>smMemRealloc()</i> – reallocate a block of memory from the shared memory system partition <i>smMemFree()</i> – free a shared memory system partition block of memory <i>smMemFindMax()</i> – find the largest free block in the shared memory system partition |
| DESCRIPTION | <p>This library provides facilities for managing the allocation of blocks of shared memory from ranges of memory called shared memory partitions. The routine <i>memPartSmCreate()</i> is used to create shared memory partitions in the shared memory pool. The created partition can be manipulated using the generic memory partition calls, <i>memPartAlloc()</i>, <i>memPartFree()</i>, etc. (for a complete list of these routines, see the manual entry for memPartLib). The maximum number of partitions that can be created is SM_OBJ_MAX_MEM_PART, defined in the configuration header file.</p> <p>The <i>smMem...()</i> routines provide an easy-to-use interface to the shared memory system partition. The shared memory system partition is created when the shared memory object facility is initialized.</p> <p>Shared memory management information and statistics display routines are provided by <i>smMemShow()</i>.</p> <p>The allocation of memory, using <i>memPartAlloc()</i> in the general case and <i>smMemMalloc()</i> for the shared memory system partition, is done with a first-fit algorithm. Adjacent blocks of memory are coalesced when freed using <i>memPartFree()</i> and <i>smMemFree()</i>.</p> <p>There is a 28-byte overhead per allocated block, and allocated blocks are aligned on a 16-byte boundary.</p> <p>All memory used by the shared memory facility must be in the same address space, that is, it must be reachable from all the CPUs with the same offset as the one used for the shared memory anchor.</p> |
| CONFIGURATION | Before routines in this library can be called, the shared memory objects facility must be initialized by a call to <i>usrSmObjInit()</i> , which is found in src/config/usrSmObj.c . This is done automatically from the root task, <i>usrRoot()</i> , in usrConfig.c , when the configuration macro INCLUDE_SM_OBJ is defined. |

ERROR OPTIONS Various debug options can be selected for each partition using *memPartOptionsSet()* and *smMemOptionsSet()*. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, options can be selected for system actions to take place when the error is detected: (1) return the error status, (2) log an error message and return the error status, or (3) log an error message and suspend the calling task.

One of the following options can be specified to determine the action to be taken when there is an attempt to allocate more memory than is available in the partition:

MEM_ALLOC_ERROR_RETURN

just return the error status to the calling task.

MEM_ALLOC_ERROR_LOG_MSG

log an error message and return the status to the calling task.

MEM_ALLOC_ERROR_LOG_AND_SUSPEND

log an error message and suspend the calling task.

The following option can be specified to check every block freed to the partition. If this option is specified, *memPartFree()* and *smMemFree()* will make a consistency check of various pointers and values in the header of the block being freed.

MEM_BLOCK_CHECK

check each block freed.

One of the following options can be specified to determine the action to be taken when a bad block is detected when freed. These options apply only if the **MEM_BLOCK_CHECK** option is selected.

MEM_BLOCK_ERROR_RETURN

just return the status to the calling task.

MEM_BLOCK_ERROR_LOG_MSG

log an error message and return the status to the calling task.

MEM_BLOCK_ERROR_LOG_AND_SUSPEND

log an error message and suspend the calling task.

The default option when a shared partition is created is **MEM_ALLOC_ERROR_LOG_MSG**.

When setting options for a partition with *memPartOptionsSet()* or *smMemOptionsSet()*, use the logical OR operator between each specified option to construct the *options* parameter. For example:

```
memPartOptionsSet (myPartId, MEM_ALLOC_ERROR_LOG_MSG |  
MEM_BLOCK_CHECK |  
MEM_BLOCK_ERROR_LOG_MSG);
```

AVAILABILITY This module is distributed as a component of the unbundled shared memory objects support option, VxMP.

INCLUDE FILES smMemLib.h

SEE ALSO **smMemShow**, **memLib**, **memPartLib**, **smObjLib**, *usrSmObjInit()*, *VxWorks Programmer's Guide: Shared Memory Objects*

smMemShow

NAME **smMemShow** – shared memory management show routines (VxMP Opt.)

ROUTINES *smMemShow()* – show the shared memory system partition blocks and statistics (VxMP Opt.)

DESCRIPTION This library provides routines to show the statistics on a shared memory system partition. General shared memory management routines are provided by **smMemLib**.

CONFIGURATION The routines in this library are included by default if **INCLUDE_SM_OBJ** is defined in **configAll.h**.

AVAILABILITY This module is distributed as a component of the unbundled shared memory objects support option, VxMP.

INCLUDE FILES **smLib.h**, **smObjLib.h**, **smMemLib.h**

SEE ALSO **smMemLib**, *VxWorks Programmer's Guide: Shared Memory Objects*

smNameLib

NAME **smNameLib** – shared memory objects name database library (VxMP Opt.)

ROUTINES *smNameAdd()* – add a name to the shared memory name database (VxMP Opt.)
smNameFind() – look up a shared memory object by name (VxMP Opt.)
smNameFindByValue() – look up a shared memory object by value (VxMP Opt.)
smNameRemove() – remove an object from the shared memory objects name database (VxMP Opt.)

DESCRIPTION This library provides facilities for managing the shared memory objects name database. The shared memory objects name database associates a name and object type with a value and makes that information available to all CPUs. A name is an arbitrary, null-terminated string. An object type is a small integer, and its value is a global (shared) ID or a global shared memory address.

Names are added to the shared memory name database with *smNameAdd()*. They are removed by *smNameRemove()*.

Objects in the database can be accessed by either name or value. The routine *smNameFind()* searches the shared memory name database for an object of a specified name. The routine *smNameFindByValue()* searches the shared memory name database for an object of a specified identifier or address.

Name database contents can be viewed using *smNameShow()*.

The maximum number of names to be entered in the database is *SM_OBJ_MAX_NAME*, defined in the configuration header file. This value is used to determine the size of a dedicated shared memory partition from which name database fields are allocated.

The estimated memory size required for the name database can be calculated as follows:

```
name database pool size = SM_OBJ_MAX_NAME * 40 (bytes)
```

The display facility for the shared memory objects name database is provided by *smNameShow*.

EXAMPLE

The following code fragment allows a task on one CPU to enter the name, associated ID, and type of a created shared semaphore into the name database. Note that CPU numbers can belong to any CPU using the shared memory objects facility.

On CPU 1:

```
#include "vxWorks.h"
#include "semLib.h"
#include "smNameLib.h"
#include "semSmLib.h"
#include "stdio.h"
testSmSem1 (void)
{
    SEM_ID smSemId;
    /* create a shared semaphore */
    if ((smSemId = semBSmCreate(SEM_Q_FIFO, SEM_EMPTY)) == NULL)
    {
        printf ("Shared semaphore creation error.");
        return (ERROR);
    }
    /*
     * make created semaphore Id available to all CPUs in
     * the system by entering its name in shared name database.
     */
    if (smNameAdd ("smSem", smSemId, T_SM_SEM_B) != OK )
    {
        printf ("Cannot add smSem into shared database.");
        return (ERROR);
    }
}
```

```
    ...
    /* now use the semaphore */
    semGive (smSemId);
    ...
}
```

On CPU 2:

```
#include "vxWorks.h"
#include "semLib.h"
#include "smNameLib.h"
#include "stdio.h"
testSmSem2 (void)
{
    SEM_ID smSemId;
    int    objType;          /* place holder for smNameFind() object type */
    /* get semaphore ID from name database */

    smNameFind ("smSem", (void **) &smSemId, &objType, WAIT_FOREVER);
    ...
    /* now that we have the shared semaphore ID, take it */

    semTake (smSemId, WAIT_FOREVER);
    ...
}
```

- CONFIGURATION** Before routines in this library can be called, the shared memory object facility must be initialized by calling *usrSmObjInit()*, which is found in **src/config/usrSmObj.c**. This is done automatically from the root task, *usrRoot()*, in **usrConfig.c** when the configuration macro **INCLUDE_SM_OBJ** is defined.
- AVAILABILITY** This module is distributed as a component of the unbundled shared memory objects support option, VxMP.
- INCLUDE FILES** **smNameLib.h**
- SEE ALSO** **smNameShow**, **smObjLib**, **smObjShow**, *usrSmObjInit()*, *VxWorks Programmer's Guide: Shared Memory Objects*

smNameShow

| | |
|----------------------|---|
| NAME | smNameShow – shared memory objects name database show routines (VxMP Opt.) |
| ROUTINES | <i>smNameShow()</i> – show the contents of the shared memory objects name database |
| DESCRIPTION | This library provides a routine to show the contents of the shared memory objects name database. The shared memory objects name database facility is provided by smNameLib . |
| CONFIGURATION | The routines in this library are included by default if INCLUDE_SM_OBJ is defined in configAll.h . |
| AVAILABILITY | This module is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| INCLUDE FILES | smNameLib.h |
| SEE ALSO | smObjLib , <i>VxWorks Programmer's Guide: Shared Memory Objects</i> |

smNetLib

| | |
|--------------------|--|
| NAME | smNetLib – VxWorks interface to the shared memory network (backplane) driver |
| ROUTINES | <i>smNetInit()</i> – initialize the shared memory network driver <i>smNetAttach()</i> – attach the shared memory network interface <i>smNetInetGet()</i> – get an address associated with a shared memory network interface |
| DESCRIPTION | <p>This library implements the VxWorks-specific portions of the shared memory network interface driver. It provides the interface between VxWorks and the network driver modules (e.g., how the OS initializes and attaches the driver, interrupt handling, etc.), as well as VxWorks-dependent system calls.</p> <p>There are three user-callable routines: <i>smNetInit()</i>, <i>smNetAttach()</i>, and <i>smNetInetGet()</i>.</p> <p>The backplane master initializes the backplane shared memory and network structures by first calling <i>smNetInit()</i>. Once the backplane has been initialized, all processors can be attached to the shared memory network via the <i>smNetAttach()</i> routine. Both <i>smNetInit()</i> and <i>smNetAttach()</i> are called automatically in usrConfig.c when backplane parameters are specified in the boot line.</p> |

The *smNetInetGet()* routine gets the Internet address associated with a backplane interface.

INCLUDE FILES *smPktLib.h*, *smUtilLib.h*

SEE ALSO *ifLib*, *if_sm*, *VxWorks Programmer's Guide: Network*

smNetShow

NAME *smNetShow* – shared memory network driver show routines

ROUTINES *smNetShow()* – show information about a shared memory network

DESCRIPTION This library provides show routines for the shared memory network interface driver. The *smNetShow()* routine is provided as a diagnostic aid to show current shared memory network status.

INCLUDE FILES *smPktLib.h*

SEE ALSO *VxWorks Programmer's Guide: Network*

smObjLib

NAME *smObjLib* – shared memory objects library (VxMP Opt.)

ROUTINES *smObjLibInit()* – install the shared memory objects facility (VxMP Opt.)
smObjSetup() – initialize the shared memory objects facility (VxMP Opt.)
smObjInit() – initialize a shared memory objects descriptor (VxMP Opt.)
smObjAttach() – attach the calling CPU to shared memory objects facility (VxMP Opt.)
smObjLocalToGlobal() – convert a local address to a global address (VxMP Opt.)
smObjGlobalToLocal() – convert a global address to a local address (VxMP Opt.)
smObjTimeoutLogEnable() – enable/disable logging of failed attempts to take a spin-lock (VxMP Opt.)

DESCRIPTION This library contains miscellaneous functions used by the shared memory objects facility. Shared memory objects provide high-speed synchronization and communication among tasks running on separate CPUs that have access to common shared memory. Shared memory objects are system objects (e.g., semaphores and message queues) that can be used across processors.

The main uses of shared memory objects are interprocessor synchronization, mutual exclusion on multiprocessor shared data structures, and high-speed data exchange.

Routines for displaying shared memory objects statistics are provided by *smObjShow()*.

SHARED MEMORY MASTER CPU

One CPU node acts as the shared memory objects master. This CPU initializes the shared memory area and sets up the shared memory anchor. These steps are performed by the master calling *smObjSetup()*. This routine should be called only once by the master CPU. Usually *smObjSetup()* is called from *usrSmObjInit()* (see "Configuration" below.)

Once *smObjSetup()* has completed successfully, there is little functional difference between the master CPU and other CPUs using shared memory objects, except that the master is responsible for maintaining the heartbeat in the shared memory header.

ATTACHING TO SHARED MEMORY

Each CPU, master or non-master, that will use shared memory objects must attach itself to the shared memory objects facility, which must already be initialized.

Before it can attach to a shared memory region, each CPU must allocate and initialize a shared memory descriptor (**SM_DESC**), which describes the individual CPU's attachment to the shared memory objects facility. Since the shared memory descriptor is used only by the local CPU, it is not necessary for the descriptor itself to be located in shared memory. In fact, it is preferable for the descriptor to be allocated from the CPU's local memory, since local memory is usually more efficiently accessed.

The shared memory descriptor is initialized by calling *smObjInit()*. This routine takes a number of parameters which specify the characteristics of the calling CPU and its access to shared memory.

Once the shared memory descriptor has been initialized, the CPU can attach itself to the shared memory region. This is done by calling *smObjAttach()*.

When *smObjAttach()* is called, it verifies that the shared memory anchor contains the value **SM_READY** and that the heartbeat located in the shared memory objects header is incrementing. If either of these conditions is not met, the routine will check periodically until either **SM_READY** or an incrementing heartbeat is recognized or a time limit is reached. The limit is expressed in seconds, and 600 seconds (10 minutes) is the default. If the time limit is reached before **SM_READY** or a heartbeat is found, **ERROR** is returned and **errno** is set to **S_smLib_DOWN**.

ADDRESS CONVERSION

This library also provides routines for converting between local and global shared memory addresses, *smObjLocalToGlobal()* and *smObjGlobalToLocal()*. A local shared memory address is the address required by the local CPU to reach a location in shared memory. A global shared memory address is a value common to all CPUs in the system used to reference a shared memory location. A global shared memory address is always an offset from the shared memory anchor.

SPIN-LOCK MECHANISM

The shared memory objects facilities use a spin-lock mechanism based on an indivisible read-modify-write (RMW) which acts as a low-level mutual exclusion device. The spin-lock mechanism is called with a system-wide parameter, **SM_OBJ_MAX_TRIES**, defined in **configAll.h**, which specifies the maximum number of RMW tries on a spin-lock location.

This parameter is set to 100 by default, but must be set to a higher value as the number of CPUs increases or when high-speed processors are used. Care must be taken that the number of RMW tries on a spin-lock on a particular CPU never reaches **SM_OBJ_MAX_TRIES**, otherwise system behavior becomes unpredictable.

The routine *smObjTimeoutLogEnable()* can be used to enable or disable the printing of a message should a shared memory object call fail while trying to take a spin-lock.

RELATION TO BACKPLANE DRIVER

Shared memory objects and the shared memory network (backplane) driver use common underlying shared memory utilities. They also use the same anchor, the same shared memory header, and the same interrupt when they are used at the same time.

LIMITATIONS

A maximum of twenty CPUs can be used concurrently with shared memory objects. Each CPU in the system must have a hardware test-and-set mechanism, which is called via the system-dependent routine *sysBusTas()*.

The use of shared memory objects raises interrupt latency, because internal mechanisms lock interrupts while manipulating critical shared data structures. Interrupt latency does not depend on the number of objects or CPUs used.

GETTING STATUS INFORMATION

The routine *smObjShow()* displays useful information regarding the current status of shared memory objects, including the number of tasks using shared objects, shared semaphores, and shared message queues, the number of names in the database, and also the maximum number of tries to get spin-lock access for the calling CPU.

CONFIGURATION

When the configuration macro **INCLUDE_SM_OBJ** is defined, the init and setup routines in this library are called automatically by *usrSmObjInit()* from the root task, *usrRoot()*, in **usrConfig.c**.

AVAILABILITY

This module is distributed as a component of the unbundled shared memory objects support option, VxMP.

INCLUDE FILES

smObjLib.h

SEE ALSO

smObjShow, **semSmLib**, **msgQSmLib**, **smMemLib**, **smNameLib**, *usrSmObjInit()*, *VxWorks Programmer's Guide: Shared Memory Objects*

smObjShow

| | |
|----------------------|--|
| NAME | smObjShow – shared memory objects show routines (VxMP Opt.) |
| ROUTINES | <i>smObjShow()</i> – display the current status of shared memory objects (VxMP Opt.) |
| DESCRIPTION | This library provides routines to show shared memory object statistics, such as the current number of shared tasks, semaphores, message queues, etc. |
| CONFIGURATION | The routines in this library are included by default if INCLUDE_SM_OBJ is defined in configAll.h . |
| AVAILABILITY | This module is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| INCLUDE FILES | smObjLib.h |
| SEE ALSO | smObjLib , <i>VxWorks Programmer's Guide: Shared Memory Objects</i> |

sn83932End

| | |
|---------------------------|---|
| NAME | sn83932End – Nat. Semi DP83932B SONIC Ethernet driver |
| ROUTINES | <i>sn83932EndLoad()</i> – initialize the driver and device |
| DESCRIPTION | <p>This module implements the National Semiconductor DP83932 SONIC Ethernet network interface driver.</p> <p>This driver is designed to be moderately generic. Thus, it operates unmodified across the range of architectures and targets supported by VxWorks. To achieve this, the driver load routine requires several target-specific parameters. The driver also depends on a few external support routines. These parameters and support routines are described below. If any of the assumptions stated below are not true for your particular hardware, this driver probably cannot function correctly with that hardware. This driver supports up to four individual units per CPU.</p> |
| BOARD LAYOUT | This device is on-board. No jumpering diagram is necessary. |
| EXTERNAL INTERFACE | This driver provides the END external interface. Thus, the only normal external interface is the <i>sn83932EndLoad()</i> routine, although <i>snEndClkEnable()</i> and <i>snEndClkDisable()</i> |

are provided for the use (optional) of the internal clock. All required parameters are passed into the load function by means of a single colon-delimited string. The *sn83932Load()* function uses *strtok()* to parse the string, which it expects to be of the following format:

```
unit_ID:devIO_addr:ivec:e_addr
```

The entry point for *sn83932EndLoad()* is defined within the **endDevTbl** in **configNet.h**.

TARGET-SPECIFIC PARAMETERS

unit_ID

A convenient holdover from the former model, this is only used in the string name for the driver.

devIO_addr

Denotes the base address of the device's I/O register set.

ivec

Denotes the interrupt vector to be used by the driver to service an interrupt from the SONIC device. The driver connects the interrupt handler to this vector by calling *intConnect()*.

e_addr

This parameter is obtained by calling *sysEnetAddrGet()*, an external support routine. It specifies the unique six-byte address assigned to the VxWorks target on the Ethernet.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires the following external support routines:

sysEnetInit()

```
void sysEnetInit (int unit)
```

This routine performs any target-specific operations that must be executed before the SONIC device is initialized. The driver calls this routine, once per unit, during the unit start-up phase.

sysEnetAddrGet()

```
STATUS sysEnetAddrGet (int unit, char *pCopy)
```

This routine provides the six-byte Ethernet address used by *unit*. It must copy the six-byte address to the space provided by *pCopy*. This routine returns OK, or ERROR if it fails. The driver calls this routine, once per unit, during the unit start-up phase.

sysEnetIntEnable()

```
void sysEnetIntEnable (int unit), void sysEnetIntDisable (int unit)
```

These routines enable or disable the interrupt from the SONIC device for the specified *unit*. Typically, this involves interrupt controller hardware, either internal or external to the CPU. The driver calls these routines only during initialization, during the unit start-up phase.

sysEnetIntAck()

void sysEnetIntAck (int unit)

This routine performs any interrupt acknowledgment or clearing that may be required. This typically involves an operation to some interrupt control hardware. The driver calls this routine from the interrupt handler.

DEVICE CONFIGURATION

Two global variables, **snEndDcr** and **snEndDcr2**, are used to set the SONIC device configuration registers. By default, the device is programmed in 32-bit mode with zero-wait states. If these values are not suitable, the **snEndDcr** and **snEndDcr2** variables should be modified before loading the driver. See the SONIC manual for information on appropriate values for these parameters.

SYSTEM RESOURCE USAGE

When implemented, this driver requires the following system resources:

- one interrupt vector
- 0 bytes in the initialized data section (data)
- 696 bytes in the uninitialized data section (BSS)

The above data and BSS requirements are for the MC68020 architecture and can vary for other architectures. Code size (text) varies greatly between architectures and is therefore not quoted here.

This driver uses *cacheDmaMalloc()* to allocate the memory to be shared with the SONIC device. The size requested is 117,188 bytes.

The SONIC device can only be operated if the shared memory region is write-coherent with the data cache. The driver cannot maintain cache coherency for the device for data that is written by the driver because fields within the shared structures are asynchronously modified by the driver and the device, and these fields may share the same cache line.

SEE ALSO

ifLib

sntpcLib

NAME

sntpcLib – Simple Network Time Protocol (SNTP) client library

ROUTINES

sntpcTimeGet() – retrieve the current time from a remote source

DESCRIPTION

This library implements the client side of the Simple Network Time Protocol (SNTP), a protocol that allows a system to maintain the accuracy of its internal clock based on time

values reported by one or more remote sources. The library is included in the VxWorks image if `INCLUDE_SNTPC` is defined at the time the image is built.

- USER INTERFACE** The *sntpTimeGet()* routine retrieves the time reported by a remote source and converts that value for POSIX-compliant clocks. The routine will either send a request and extract the time from the reply, or it will wait until a message is received from an SNTP/NTP server executing in broadcast mode.
- INCLUDE FILES** `sntpLib.h`
- SEE ALSO** `clockLib`, RFC 1769

sntpLib

- NAME** `sntpLib` – Simple Network Time Protocol (SNTP) server library
- ROUTINES** *sntpClockSet()* – assign a routine to access the reference clock
sntpNsecToFraction() – convert portions of a second to NTP format
sntpConfigSet() – change SNTP server broadcast settings
- DESCRIPTION** This library implements the server side of the Simple Network Time Protocol (SNTP), a protocol that allows a system to maintain the accuracy of its internal clock based on time values reported by one or more remote sources. The library is included in the VxWorks image if `INCLUDE_SNTPS` is defined at the time the image is built.
- USER INTERFACE** The routine *sntpInit()* is called automatically during system startup when the SNTP server library is included in the VxWorks image. Depending on the value of `SNTPS_MODE`, the server executes in either a passive or an active mode. When `SNTPS_MODE` is set to `SNTP_PASSIVE` (0x2), the server waits for requests from clients, and sends replies containing an NTP timestamp. When the mode is set to `SNTP_ACTIVE` (0x1), the server transmits NTP timestamp information at fixed intervals.
- When executing in active mode, the SNTP server uses the `SNTPS_DSTADDR` and `SNTPS_INTERVAL` definitions to determine the target IP address and broadcast interval. By default, the server will transmit the timestamp information to the local subnet broadcast address every 64 seconds. These settings can be changed with a call to the *sntpConfigSet()* routine. The SNTP server operating in active mode will still respond to client requests.
- The `SNTP_PORT` definition in assigns the source and destination UDP port. The default port setting is 123 as specified by the relevant RFC. Finally, the SNTP server requires access to a reliable external time source. The `SNTPS_TIME_HOOK` constant specifies the name of a routine with the following interface:

sockLib

```
STATUS sntpTimeHook (int request, void *pBuffer);
```

This routine can be assigned directly by altering the value of `SNTPS_TIME_HOOK` or can be installed by a call to the `sntpClockSet()` routine. The manual pages for `sntpClockSet()` describe the parameters and required operation of the timestamp retrieval routine. Until this routine is specified, the SNTP server will not provide timestamp information.

INCLUDE FILES **sntpsLib.h**

SEE ALSO **sntpcLib**, RFC 1769

sockLib

NAME **sockLib** – generic socket library

ROUTINES

- `socket()` – open a socket
- `bind()` – bind a name to a socket
- `listen()` – enable connections to a socket
- `accept()` – accept a connection from a socket
- `connect()` – initiate a connection to a socket
- `connectWithTimeout()` – try to connect over a socket for a specified duration
- `sendto()` – send a message to a socket
- `send()` – send data to a socket
- `sendmsg()` – send a message to a socket
- `recvfrom()` – receive a message from a socket
- `recv()` – receive data from a socket
- `recvmsg()` – receive a message from a socket
- `setsockopt()` – set socket options
- `getsockopt()` – get socket options
- `getsockname()` – get a socket name
- `getpeername()` – get the name of a connected peer
- `shutdown()` – shut down a network connection

DESCRIPTION This library provides UNIX BSD 4.4 compatible socket calls. Use these calls to open, close, read, and write sockets. These sockets can join processes on the same CPU or on different CPUs between which there is a network connection. The calling sequences of these routines are identical to their equivalents under UNIX BSD 4.4.

ADDRESS FAMILY VxWorks sockets support only the Internet Domain address family. Use `AF_INET` for the `domain` argument in subroutines that require it. There is no support for the UNIX Domain address family.

IOCTL FUNCTIONS Sockets respond to the following *ioctl()* functions. These functions are defined in the header files **ioLib.h** and **ioctl.h**.

FIONBIO

Turns on/off non-blocking I/O.

```
on = TRUE;
status = ioctl (sFd, FIONBIO, &on);
```

FIONREAD

Reports the number of read-ready bytes available on the socket. On the return of *ioctl()*, *bytesAvailable* has the number of bytes available to read from the socket.

```
status = ioctl (sFd, FIONREAD, &bytesAvailable);
```

SIOCATMARK

Reports whether there is out-of-band data to be read from the socket. On the return of *ioctl()*, *atMark* is TRUE (1) if there is out-of-band data. Otherwise, it is FALSE (0).

```
status = ioctl (sFd, SIOCATMARK, &atMark);
```

INCLUDE FILES **types.h, mbuf.h, socket.h, socketvar.h**

SEE ALSO **netLib**, *VxWorks Programmer's Guide: Network*

spyLib

NAME **spyLib** – spy CPU activity library

ROUTINES *spyLibInit()* – initialize task cpu utilization tool package

DESCRIPTION This library provides a facility to monitor tasks' use of the CPU. The primary interface routine, *spy()*, periodically calls *spyReport()* to display the amount of CPU time utilized by each task, the amount of time spent at interrupt level, the amount of time spent in the kernel, and the amount of idle time. It also displays the total usage since the start of *spy()* (or the last call to *spyClkStart()*), and the change in usage since the last *spyReport()*.

CPU usage can also be monitored manually by calling *spyClkStart()* and *spyReport()*, instead of *spy()*. In this case, *spyReport()* provides a one-time report of the same information provided by *spy()*.

Data is gathered by an interrupt-level routine that is connected by *spyClkStart()* to the auxiliary clock. Currently, this facility cannot be used with CPUs that have no auxiliary clock. Interrupts that are at a higher level than the auxiliary clock's interrupt level cannot be monitored.

All user interface routine except *spyLibInit()* are available through **usrLib**.

EXAMPLE

The following call:

```
-> spy 10, 200
```

will generate a report in the following format every 10 seconds, gathering data at the rate of 200 times per second.

| NAME | ENTRY | TID | PRI | total % (ticks) | delta % (ticks) |
|--------------|------------|-------|-----|-----------------|-----------------|
| tExcTask | _excTask | fbb58 | 0 | 0% (0) | 0% (0) |
| tLogTask | _logTask | fa6e0 | 0 | 0% (0) | 0% (0) |
| tShell | _shell | e28a8 | 1 | 0% (4) | 0% (0) |
| tRlogind | _rlogind | f08dc | 2 | 0% (0) | 0% (0) |
| tRlogOutTask | _rlogOutTa | e93e0 | 2 | 2% (173) | 2% (46) |
| tRlogInTask | _rlogInTas | e7f10 | 2 | 0% (0) | 0% (0) |
| tSpyTask | _spyTask | ffe9c | 5 | 1% (116) | 1% (28) |
| tNetTask | _netTask | f3e2c | 50 | 0% (4) | 0% (1) |
| tPortmapd | _portmapd | ef240 | 100 | 0% (0) | 0% (0) |
| KERNEL | | | | 1% (105) | 0% (10) |
| INTERRUPT | | | | 0% (0) | 0% (0) |
| IDLE | | | | 95% (7990) | 95% (1998) |
| TOTAL | | | | 99% (8337) | 98% (2083) |

The "total" column reflects CPU activity since the initial call to *spy()* or the last call to *spyClkStart()*. The "delta" column reflects activity since the previous report. A call to *spyReport()* will produce a single report; however, the initial auxiliary clock interrupts and data collection must first be started using *spyClkStart()*.

Data collection/clock interrupts and periodic reporting are stopped by calling:

```
-> spyStop
```

INCLUDE FILES **spyLib.h**

SEE ALSO **usrLib**

sramDrv

NAME **sramDrv** – PCMCIA SRAM device driver

ROUTINES *sramDrv()* – install a PCMCIA SRAM memory driver
sramMap() – map PCMCIA memory onto a specified ISA address space
sramDevCreate() – create a PCMCIA memory disk device

DESCRIPTION This is a device driver for the SRAM PC card. The memory location and size are specified when the "disk" is created.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. However, two routines must be called directly: *sramDrv()* to initialize the driver, and *sramDevCreate()* to create block devices. Additionally, the *sramMap()* routine is called directly to map the PCMCIA memory onto the ISA address space. Note that this routine does not use any mutual exclusion or synchronization mechanism; thus, special care must be taken in the multitasking environment.

Before using this driver, it must be initialized by calling *sramDrv()*. This routine should be called only once, before any reads, writes, or calls to *sramDevCreate()* or *sramMap()*. It can be called from *usrRoot()* in *usrConfig.cor* at some later point.

SEE ALSO *VxWorks Programmer's Guide: I/O System*

st16552Sio

NAME *st16552Sio* – ST 16C552 DUART tty driver

ROUTINES *st16552DevInit()* – initialise an ST16552 channel
st16552IntWr() – handle a transmitter interrupt
st16552IntRd() – handle a receiver interrupt
st16552IntEx() – miscellaneous interrupt processing
st16552Int() – interrupt level processing
st16552MuxInt() – multiplexed interrupt level processing

DESCRIPTION This is the device driver for the Startech ST16C552 DUART, similar, but not quite identical to the National Semiconductor 16550 UART.

The chip is a dual universal asynchronous receiver/transmitter with 16 byte transmit and receive FIFOs and a programmable baud-rate generator. Full modem control capability is included and control over the four interrupts that can be generated: Tx, Rx, Line status, and modem status. Only the Rx and Tx interrupts are used by this driver. The FIFOs are enabled for both Tx and Rx by this driver.

Only asynchronous serial operation is supported by the UART which supports 5 to 8 bit word lengths with or without parity and with one or two stop bits. The only serial word format supported by the driver is 8 data bits, 1 stop bit, no parity. The default baud rate is determined by the BSP by filling in the *ST16552_CHAN* structure before calling *ambaDevInit()*.

The exact baud rates supported by this driver will depend on the crystal fitted (and consequently the input clock to the baud-rate generator), but in general, baud rates from about 50 to about 115200 are possible.

DATA STRUCTURES

An **ST16552_CHAN** data structure is used to describe the two channels of the chip and, if necessary, an **ST16552_MUX** structure is used to describe the multiplexing of the interrupts for the two channels of the DUART. These structures are described in **h/drv/sio/ambaSio.h**.

CALLBACKS

Servicing a "transmitter ready" interrupt involves making a callback to a higher level library in order to get a character to transmit. By default, this driver installs dummy callback routines which do nothing. A higher layer library that wants to use this driver (e.g. **ttYDrv**) will install its own callback routine using the **SIO_INSTALL_CALLBACK** ioctl command. Likewise, a receiver interrupt handler makes a callback to pass the character to the higher layer library.

MODES

This driver supports both polled and interrupt modes.

USAGE

The driver is typically called only by the BSP. This module's directly callable routines are **st16552DevInit()**, **st16552Int()**, **st16552IntRd()**, **st16552IntWr()**, and **st16552MuxInt()**.

The BSP's **sysHwInit()** routine typically calls **sysSerialHwInit()**, which initialises all the hardware-specific values in the **ST16552_CHAN** structure before calling **st16552DevInit()** which resets the device and installs the driver function pointers. After this the UART will be enabled and ready to generate interrupts, but those interrupts will be disabled in the interrupt controller.

The following example shows the first parts of the initialization:

```
#include "drv/sio/st16552Sio.h"
LOCAL ST16552_CHAN st16552Chan[N_16552_CHANNELS];
void sysSerialHwInit (void)
{
    int i;
    for (i = 0; i < N_16552_CHANNELS; i++)
    {
        st16552Chan[i].regDelta = devParas[i].regSpace;
        st16552Chan[i].regs = devParas[i].baseAdrs;
        st16552Chan[i].baudRate = CONSOLE_BAUD_RATE;
        st16552Chan[i].xtal = UART_XTAL_FREQ;
        st16552Chan[i].level = devParas[i].intLevel;
        /*
         * Initialise driver functions, getTxChar, putRcvChar and
         * channelMode and init UART.
         */
        st16552DevInit(&st16552Chan[i]);
    }
}
```

The BSP's *sysHwInit2()* routine typically calls *sysSerialHwInit2()*, which connects the chips interrupts via *intConnect()* (either the single interrupt *st16552Int*, the three interrupts *st16552IntWr*, *st16552IntRd*, and *st16552IntEx*, or the multiplexed interrupt handler *st16552MuxInt* which will cope with both channels of a DUART producing the same interrupt). It then enables those interrupts in the interrupt controller as shown in the following example:

```
void sysSerialHwInit2 (void)
{
    /* Connect the multiplexed interrupt handler */
    (void) intConnect (INUM_TO_IVEC(devParas[0].vector),
                     st16552MuxInt, (int) &st16552Mux);
    intEnable (devParas[0].intLevel);
}
```

BSP

By convention all the BSP-specific serial initialisation is performed in a file called *sysSerial.c*, which is #include'd by *sysLib.c*. *sysSerial.c* implements at least four functions, *sysSerialHwInit()*, *sysSerialHwInit2()*, *sysSerialChanGet()*, and *sysSerialReset()*. The first two have been described above, the others work as follows:

sysSerialChanGet() is called by *usrRoot()* to get the serial channel descriptor associated with a serial channel number. The routine takes a single parameter which is a channel number ranging between zero and *NUM_TTY*. It returns a pointer to the corresponding channel descriptor, *SIO_CHAN **, which is just the address of the *ST16552_CHAN* structure.

sysSerialReset() is called from *sysToMonitor()* and should reset the serial devices to an inactive state (prevent them from generating any interrupts).

INCLUDE FILES *drv/sio/st16552Sio.h*, *sioLib.h*

SEE ALSO *Startech ST16C552 Data Sheet*

subagentLib

NAME

subagentLib – encode, decode, and process agent and subagent messages

ROUTINES

snmpSubEncode() – encode a packet for transmission to master agent or subagent
snmpSaHandlerAsync() – asynchronous message processing routine for the subagent
snmpSaHandlerWR() – provide *snmpSaHandlerAsync()* functionality synchronously
snmpSaHandlerContinue() – subagent continuation function
snmpSaHandlerFinish() – encode packet for subagent I/O completion

symLib

snmpSaHandlerCleanup() – cleanup routine for subagent
snmpMasterHandlerAsync() – process messages from the subagent asynchronously
snmpMasterHandlerWR() – synchronous version of *snmpMasterHandlerAsync()*
snmpMasterQueryHandler() – handles replies from the subagent
snmpMasterCleanup() – free up resources after a query times out

DESCRIPTION This module provides the core routines for processing the messages passed between the SNMP master agent and its subagents. Thus, this library includes routines for encoding and decoding a package. It also includes the routines used to sort the messages according to type and then respond to each specific message appropriately.

symLib

NAME **symLib** – symbol table subroutine library

ROUTINES

- symLibInit()* – initialize the symbol table library
- symTblCreate()* – create a symbol table
- symTblDelete()* – delete a symbol table
- symAdd()* – create and add a symbol to a symbol table, including a group number
- symRemove()* – remove a symbol from a symbol table
- symFindByName()* – look up a symbol by name
- symFindByNameAndType()* – look up a symbol by name and type
- symFindByValue()* – look up a symbol by value
- symFindByValueAndType()* – look up a symbol by value and type
- symEach()* – call a routine to examine each entry in a symbol table

DESCRIPTION This library provides facilities for managing symbol tables. A symbol table associates a name and type with a value. A name is simply an arbitrary, null-terminated string. A symbol type is a small integer (typedef **SYM_TYPE**), and its value is a character pointer. Though commonly used as the basis for object loaders, symbol tables may be used whenever efficient association of a value with a name is needed.

If you use the **symLib** subroutines to manage symbol tables local to your own applications, the values for **SYM_TYPE** objects are completely arbitrary; you can use whatever one-byte integers are appropriate for your application.

If you use the **symLib** subroutines to manipulate the VxWorks system symbol table (whose ID is recorded in the global **sysSymTbl**), the values for **SYM_TYPE** are **N_ABS**, **N_TEXT**, **N_DATA**, and **N_BSS** (defined in **a_out.h**); these are all even numbers, and any of them may be combined (via boolean or) with **N_EXT** (1). These values originate in the section names for a.out object code format, but the VxWorks system symbol table uses them as symbol types across all object formats. (The VxWorks system symbol table also occasionally includes additional types, in some object formats.)

Tables are created with *symTblCreate()*, which returns a symbol table ID. This ID serves as a handle for symbol table operations, including the adding to, removing from, and searching of tables. All operations on a symbol table are interlocked by means of a mutual-exclusion semaphore in the symbol table structure. Tables are deleted with *symTblDelete()*.

Symbols are added to a symbol table with *symAdd()*. Each symbol has a name, a value, and a type. Symbols are removed from a symbol table with *symRemove()*.

Symbols can be accessed by either name or value. The routine *symFindByName()* searches the symbol table for a symbol of a specified name. The routine *symFindByValue()* finds the symbol with the value closest to a specified value. The routines *symFindByNameAndType()* and *symFindByValueAndType()* allow the symbol type to be used as an additional criterion in the searches.

Symbols in the symbol table are hashed by name into a hash table for fast look-up by name, e.g., by *symFindByName()*. The size of the hash table is specified during the creation of a symbol table. Look-ups by value, e.g., *symFindByValue()*, must search the table linearly; these look-ups can thus be much slower.

The routine *symEach()* allows each symbol in the symbol table to be examined by a user-specified function.

Name clashes occur when a symbol added to a table is identical in name and type to a previously added symbol. Whether or not symbol tables can accept name clashes is set by a parameter when the symbol table is created with *symTblCreate()*. If name clashes are not allowed, *symAdd()* will return an error if there is an attempt to add a symbol with identical name and type. If name clashes are allowed, adding multiple symbols with the same name and type will be permitted. In such cases, *symFindByName()* will return the value most recently added, although all versions of the symbol can be found by *symEach()*.

INCLUDE FILES **symLib.h**

SEE ALSO **loadLib**

symSyncLib

| | |
|--------------------|--|
| NAME | symSyncLib – host/target symbol table synchronization |
| ROUTINES | <i>symSyncLibInit()</i> – initialize host/target symbol table synchronization <i>symSyncTimeoutSet()</i> – set WTX timeout <i>syncTgtSafeModCheck()</i> – check if a target module can be safely used |
| DESCRIPTION | <p>This module provides host/target symbol table synchronization. With synchronization, every module or symbol added to the run-time system from either the target or host side can be seen by facilities on both the target and the host. Symbol-table synchronization makes it possible to use host tools to debug application modules loaded with the target loader or from a target file system. To enable synchronization, two actions must be performed:</p> <ol style="list-style-type: none">1 The module is initialized by <i>symSyncLibInit()</i>, which is called automatically when the configuration macro INCLUDE_SYM_TBL_SYNC is defined.2 The target server is launched with the -s option. |

If synchronization is enabled, **symSyncLib** spawns a synchronization task on the target, **tSymSync**. This task behaves as a WTX tool and attaches itself to the target server. When the task starts, it synchronizes target and host symbol tables so that every module loaded on the target before the target server was started can be seen by the host tools. This feature is particularly useful if VxWorks is started with a target-based startup script before the target server has been launched.

The **tSymSync** task also assures synchronization as new symbols are added by either the target or the host tools. The task waits for synchronization events on two channels: a WTX event from the host or a message queue addition from the target.

The **tSymSync** task, like all WTX tools, must be able to connect to the WTX registry. To make the WTX registry accessible from the target, do one of the following:

- 1 Boot the target from a host on the same subnet as the registry.
- 2 Start the registry on the same host the target boots from.
- 3 Add the needed routes with *routeAdd()* calls, possibly in a startup script.

Neither the host tools nor the target loader wait for synchronization completion to return. To know when the synchronization is complete, you can wait for the corresponding event sent by the target server, or, if your target server was started with the **-V** option, it prints a message indicating synchronization has been completed.

The event sent by the target server is of the following format:

```
SYNC_DONE syncType syncObj syncStatus
```


The following are examples of messages displayed by the target server indicating synchronization is complete:

```
Added target_modules      to target-server.....done
Added ttTest.o.68k       to target.....done
```

If synchronization fails, the following message is displayed:

```
Added gopher.o           to target.....failed
```

This error generally means that synchronization of the corresponding module or symbol is no longer possible because it no longer exists in the original symbol table. If so, it will be followed by:

```
Removed gopher.o        from target.....failed
```

Failure can also occur if a timeout is reached. Call *symSyncTimeoutSet()* to modify the WTX timeout between the target synchronization task and the target server.

LIMITATIONS

Hardware: Because the synchronization task uses the WTX protocol to communicate with the target server, the target must include network facilities. Depending on how much synchronization is to be done (number of symbols to transfer), a reasonable throughput between the target server and target agent is required (the wdbRPC backend is recommended when large modules are to be loaded).

Performance: The synchronization task requires some minor overhead in target routines *msgQSend()*, *loadModule()*, *symAdd()*, and *symRemove()*; however, if an application sends more than 15 synchronization events, it will fill the message queue and then need to wait for a synchronization event to be processed by **tSymSync**. Also, waiting for host synchronization events is done by polling; thus there may be some impact on performance if there are lower-priority tasks than **tSymSync**. If no more synchronization is needed, **tSymSync** can be suspended.

Known problem: Modules with undefined symbols that are loaded from the target are not synchronized; however, they are synchronized if they are loaded from the host.

SEE ALSO

tgtsvr

sysLib

| | |
|--------------------|--|
| NAME | sysLib – system-dependent library |
| ROUTINES | <p><i>sysClkConnect()</i> – connect a routine to the system clock interrupt <i>sysClkDisable()</i> – turn off system clock interrupts <i>sysClkEnable()</i> – turn on system clock interrupts <i>sysClkRateGet()</i> – get the system clock rate <i>sysClkRateSet()</i> – set the system clock rate <i>sysAuxClkConnect()</i> – connect a routine to the auxiliary clock interrupt <i>sysAuxClkDisable()</i> – turn off auxiliary clock interrupts <i>sysAuxClkEnable()</i> – turn on auxiliary clock interrupts <i>sysAuxClkRateGet()</i> – get the auxiliary clock rate <i>sysAuxClkRateSet()</i> – set the auxiliary clock rate <i>sysIntDisable()</i> – disable a bus interrupt level <i>sysIntEnable()</i> – enable a bus interrupt level <i>sysBusIntAck()</i> – acknowledge a bus interrupt <i>sysBusIntGen()</i> – generate a bus interrupt <i>sysMailboxConnect()</i> – connect a routine to the mailbox interrupt <i>sysMailboxEnable()</i> – enable the mailbox interrupt <i>sysNvRamGet()</i> – get the contents of non-volatile RAM <i>sysNvRamSet()</i> – write to non-volatile RAM <i>sysModel()</i> – return the model name of the CPU board <i>sysBspRev()</i> – return the BSP version and revision number <i>sysHwInit()</i> – initialize the system hardware <i>sysPhysMemTop()</i> – get the address of the top of memory <i>sysMemTop()</i> – get the address of the top of logical memory <i>sysToMonitor()</i> – transfer control to the ROM monitor <i>sysProcNumGet()</i> – get the processor number <i>sysProcNumSet()</i> – set the processor number <i>sysBusTas()</i> – test and set a location across the bus <i>sysScsiBusReset()</i> – assert the RST line on the SCSI bus (Western Digital WD33C93 only) <i>sysScsiInit()</i> – initialize an on-board SCSI port <i>sysScsiConfig()</i> – system SCSI configuration <i>sysLocalToBusAdrs()</i> – convert a local address to a bus address <i>sysBusToLocalAdrs()</i> – convert a bus address to a local address <i>sysSerialHwInit()</i> – initialize the BSP serial devices to a quiescent state <i>sysSerialHwInit2()</i> – connect BSP serial device interrupts <i>sysSerialReset()</i> – reset all SIO devices to a quiet state <i>sysSerialChanGet()</i> – get the SIO_CHAN device associated with a serial channel</p> |
| DESCRIPTION | This library provides board-specific routines. |

NOTE: This is a generic reference entry for a BSP-specific library; this description contains general information only. For features and capabilities specific to the system library included in your BSP, see your BSP's reference entry for **sysLib**.

The file **sysLib.c** provides the board-level interface on which VxWorks and application code can be built in a hardware-independent manner. The functions addressed in this file include:

Initialization functions

- initialize the hardware to a known state
- identify the system
- initialize drivers, such as SCSI or custom drivers

Memory/address space functions

- get the on-board memory size
- make on-board memory accessible to external bus
- map local and bus address spaces
- enable/disable cache memory
- set/get nonvolatile RAM (NVRAM)
- define board's memory map (optional)
- virtual-to-physical memory map declarations for processors with MMUs

Bus interrupt functions

- enable/disable bus interrupt levels
- generate bus interrupts

Clock/timer functions

- enable/disable timer interrupts
- set the periodic rate of the timer

Mailbox/location monitor functions

- enable mailbox/location monitor interrupts for VME-based boards

The **sysLib** library does not support every feature of every board; a particular board may have various extensions to the capabilities described here. Conversely, some boards do not support every function provided by this library. Some boards provide some of the functions of this library by means of hardware switches, jumpers, or PALs, instead of software-controllable registers.

Typically, most functions in this library are not called by the user application directly. The configuration modules **usrConfig.c** and **bootConfig.c** are responsible for invoking the routines at the appropriate time. Device drivers may use some of the memory mapping routines and bus functions.

INCLUDE FILES **sysLib.h**

SEE ALSO *VxWorks Programmer's Guide: Configuration and Build*, BSP-specific reference entry for **sysLib**

tapeFsLib

| | |
|--------------------|--|
| NAME | tapeFsLib – tape sequential device file system library |
| ROUTINES | <i>tapeFsDevInit()</i> – associate a sequential device with tape volume functions <i>tapeFsInit()</i> – initialize the tape volume library <i>tapeFsReadyChange()</i> – notify tapeFsLib of a change in ready status <i>tapeFsVolUnmount()</i> – disable a tape device volume |
| DESCRIPTION | This library provides basic services for tape devices that do not use a standard file or directory structure on tape. The tape volume is treated much like a large file. The tape may either be read or written. However, there is no high-level organization of the tape into files or directories, which must be provided by a higher-level layer. |

USING THIS LIBRARY

The various routines provided by the VxWorks tape file system, or **tapeFs**, can be categorized into three broad groupings: general initialization, device initialization, and file system operation.

The *tapeFsInit()* routine is the principal general initialization function; it needs to be called only once, regardless of how many **tapeFs** devices are used.

To initialize devices, *tapeFsDevInit()* must be called for each **tapeFs** device.

Use of this library typically occurs through standard use of the I/O system routines *open()*, *close()*, *read()*, *write()* and *ioctl()*. Besides these standard I/O system operations, several routines are provided to inform the file system of changes in the system environment. The *tapeFsVolUnmount()* routine informs the file system that a particular device should be unmounted; any synchronization should be done prior to invocation of this routine, in preparation for a tape volume change. The *tapeFsReadyChange()* routine is used to inform the file system that a tape may have been swapped and that the next tape operation should first remount the tape. Information about a ready-change is also obtained from the driver using the **SEQ_DEV** device structure. Note that *tapeFsVolUnmount()* and *tapeFsReadyChange()* should be called only after a file has been closed.

INITIALIZATION OF THE FILE SYSTEM

Before any other routines in **tapeFsLib** can be used, *tapeFsInit()* must be called to initialize the library. This implementation of the tape file system assumes only one file descriptor per volume. However, this constraint can be changed in case a future implementation demands multiple file descriptors per volume.

During the *tapeFsInit()* call, the tape device library is installed as a driver in the I/O system driver table. The driver number associated with it is then placed in a global variable, **tapeFsDrvNum**.

To enable this initialization, define `INCLUDE_TAPEFS` in the BSP, or simply start using the tape file system with a call to `tapeFsDevInit()` and `tapeFsInit()` will be called automatically if it has not been called before.

DEFINING A TAPE DEVICE

To use this library for a particular device, the device structure used by the device driver must contain, as the very first item, a sequential device description structure (`SEQ_DEV`). The `SEQ_DEV` must be initialized before calling `tapeFsDevInit()`. The driver places in the `SEQ_DEV` structure the addresses of routines that it must supply: one that reads one or more blocks, one that writes one or more blocks, one that performs I/O control (`ioctl()`) on the device, one that writes file marks on a tape, one that rewinds the tape volume, one that reserves a tape device for use, one that releases a tape device after use, one that mounts/unmounts a volume, one that spaces forward or backwards by blocks or file marks, one that erases the tape, one that resets the tape device, and one that checks the status of the device. The `SEQ_DEV` structure also contains fields that describe the physical configuration of the device. For more information about defining sequential devices, see the *VxWorks Programmer's Guide: I/O System*.

INITIALIZATION OF THE DEVICE

The `tapeFsDevInit()` routine is used to associate a device with the `tapeFsLib` functions. The `volName` parameter expected by `tapeFsDevInit()` is a pointer to a name string which identifies the device. This string serves as the pathname for I/O operations which operate on the device and appears in the I/O system device table, which can be displayed using `iosDevShow()`.

The `pSeqDev` parameter expected by `tapeFsDevInit()` is a pointer to the `SEQ_DEV` structure describing the device and containing the addresses of the required driver functions.

The `pTapeConfig` parameter is a pointer to a `TAPE_CONFIG` structure that contains information specifying how the tape device should be configured. The configuration items are fixed/variable block size, rewind/no-rewind device, and number of file marks to be written. For more information about the `TAPE_CONFIG` structure, look at the header file `tapeFsLib.h`.

The syntax of the `tapeFsDevInit()` routine is as follows:

```
tapeFsDevInit
(
    char *      volName,      /* name to be used for volume */
    SEQ_DEV *   pSeqDev,     /* pointer to device descriptor */
    TAPE_CONFIG * pTapeConfig /* pointer to tape config info */
)
```

When `tapeFsLib` receives a request from the I/O system, after `tapeFsDevInit()` has been called, it calls the device driver routines (whose addresses were passed in the `SEQ_DEV` structure) to access the device.

OPENING AND CLOSING A FILE

A tape volume is opened by calling the I/O system routine *open()*. A file can be opened only with the **O_RDONLY** or **O_WRONLY** flags. The **O_RDWR** mode is not used by this library. A call to *open()* initializes the file descriptor buffer and state information, reserves the tape device, rewinds the tape device if it was configured as a rewind device, and mounts a volume. Once a tape volume has been opened, that tape device is reserved, disallowing any other system from accessing that device until the tape volume is closed. Also, the single file descriptor is marked "in use" until the file is closed, making sure that a file descriptor is not opened multiple times.

A tape device is closed by calling the I/O system routine *close()*. Upon a *close()* request, any unwritten buffers are flushed, the device is rewound (if it is a rewind device), and, finally, the device is released.

UNMOUNTING VOLUMES (CHANGING TAPES)

A tape volume should be unmounted before it is removed. When unmounting a volume, make sure that any open file is closed first. A tape may be unmounted by calling *tapeFsVolUnmount()* directly.

If a file is open, it is not correct to change the medium and continue with the same file descriptor still open. Since tapeFs assumes only one file descriptor per device, to reuse that device, the file must be closed and opened later for the new tape volume.

Before *tapeFsVolUnmount()* is called, the device should be synchronized by invoking the *ioctl()* **FIOSYNC** or **FIOFLUSH**. It is the responsibility of the higher-level layer to synchronize the tape file system before unmounting. Failure to synchronize the volume before unmounting may result in loss of data.

IOCTL FUNCTIONS The VxWorks tape sequential device file system supports the following *ioctl()* functions. The functions listed are defined in the header files **ioLib.h** and **tapeFsLib.h**.

FIOFLUSH

Writes all modified file descriptor buffers to the physical device.

```
status = ioctl (fd, FIOFLUSH, 0);
```

FIOSYNC

Performs the same function as **FIOFLUSH**.

FIOBLKSIZEGET

Returns the value of the block size set on the physical device. This value is compared against the **sd_blkSize** value set in the **SEQ_DEV** device structure.

FIOBLKSIZESET

Sets a specified block size value on the physical device and also updates the value in the **SEQ_DEV** and **TAPE_VOL_DESC** structures, unless the supplied value is zero, in which case the device structures are updated but the device is not set to zero. This is because zero implies variable block operations, therefore the device block size is ignored.

MTIOCTOP

Allows use of the standard UNIX MTIO `ioctl` operations by means of the MTOP structure. The MTOP structure appears as follows:

```
typedef struct mtop
{
    short      mt_op;           /* operation */
    int        mt_count;       /* number of operations */
} MTOP;
```

Use these `ioctl()` operations as follows:

```
MTOP mtop;
mtop.mt_op = MTWEOF;
mtop.mt_count = 1;
status = ioctl (fd, MTIOCTOP, (int) &mtop);
```

The permissible values for `mt_op` are:

MTWEOF

Writes an end-of-file record to tape. An end-of-file record is a file mark.

MTFSF

Forward space over a file mark and position the tape head in the gap between the file mark just skipped and the next data block. Any buffered data is flushed out to the tape if the tape is in write mode.

MTBSF

Backward space over a file mark and position the tape head in the gap preceding the file mark, that is, right before the file mark. Any buffered data is flushed out to the tape if the tape is in write mode.

MTFSR

Forward space over a data block and position the tape head in the gap between the block just skipped and the next block. Any buffered data is flushed out to the tape if the tape is in write mode.

MTBSR

Backward space over a data block and position the tape head right before the block just skipped. Any buffered data is flushed out to the tape if the tape is in write mode.

MTREW

Rewind the tape to the beginning of the medium. Any buffered data is flushed out to the tape if the tape is in write mode.

MTOFFL

Rewind and unload the tape. Any buffered data is flushed out to the tape if the tape is in write mode.

MTNOP

No operation, but check the status of the device, thus setting the appropriate `SEQ_DEV` fields.

taskArchLib

MTRETEN

Retension the tape. This command usually sets tape tension and can be used in either read or write mode. Any buffered data is flushed out to tape if the tape is in write mode.

MTERASE

Erase the entire tape and rewind it.

MTEOM

Position the tape at the end of the medium and unload the tape. Any buffered data is flushed out to the tape if the tape is in write mode.

INCLUDE FILES **tapeFsLib.h**

SEE ALSO **ioLib, iosLib**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

taskArchLib

NAME **taskArchLib** – architecture-specific task management routines

ROUTINES *taskSRSet()* – set the task status register (MC680x0, MIPS, i386/i486)
taskSRInit() – initialize the default task status register (MIPS)

DESCRIPTION This library provides architecture-specific task management routines that set and examine architecture-dependent registers. For information about architecture-independent task management facilities, see the manual entry for **taskLib**.

NOTE There are no application-level routines in **taskArchLib** for SPARC.

INCLUDE FILES **regs.h, taskArchLib.h**

SEE ALSO **taskLib**

taskHookLib

NAME **taskHookLib** – task hook library

ROUTINES *taskHookInit()* – initialize task hook facilities
taskCreateHookAdd() – add a routine to be called at every task create
taskCreateHookDelete() – delete a previously added task create routine

taskSwitchHookAdd() – add a routine to be called at every task switch
taskSwitchHookDelete() – delete a previously added task switch routine
taskDeleteHookAdd() – add a routine to be called at every task delete
taskDeleteHookDelete() – delete a previously added task delete routine

DESCRIPTION

This library provides routines for adding extensions to the VxWorks tasking facility. To allow task-related facilities to be added to the system without modifying the kernel, the kernel provides call-outs every time a task is created, switched, or deleted. The call-outs allow additional routines, or "hooks," to be invoked whenever these events occur. The hook management routines below allow hooks to be dynamically added to and deleted from the current lists of create, switch, and delete hooks:

taskCreateHookAdd() and *taskCreateHookDelete()*

Add and delete routines to be called when a task is created.

taskSwitchHookAdd() and *taskSwitchHookDelete()*

Add and delete routines to be called when a task is switched.

taskDeleteHookAdd() and *taskDeleteHookDelete()*

Add and delete routines to be called when a task is deleted.

This facility is used by **dbgLib** to provide task-specific breakpoints and single-stepping. It is used by **taskVarLib** for the "task variable" mechanism. It is also used by **fppLib** for floating-point coprocessor support.

NOTE

It is possible to have dependencies among task hook routines. For example, a delete hook may use facilities that are cleaned up and deleted by another delete hook. In such cases, the order in which the hooks run is important. VxWorks runs the create and switch hooks in the order in which they were added, and runs the delete hooks in reverse of the order in which they were added. Thus, if the hooks are added in "hierarchical" order, such that they rely only on facilities whose hook routines have already been added, then the required facilities will be initialized before any other facilities need them, and will be deleted after all facilities are finished with them.

VxWorks facilities guarantee this by having each facility's initialization routine first call any prerequisite facility's initialization routine before adding its own hooks. Thus, the hooks are always added in the correct order. Each initialization routine protects itself from multiple invocations, allowing only the first invocation to have any effect.

INCLUDE FILES

taskHookLib.h

SEE ALSO

dbgLib, **fppLib**, **taskLib**, **taskVarLib**, *VxWorks Programmer's Guide: Basic OS*

taskHookShow

| | |
|----------------------|---|
| NAME | taskHookShow – task hook show routines |
| ROUTINES | <i>taskHookShowInit()</i> – initialize the task hook show facility <i>taskCreateHookShow()</i> – show the list of task create routines <i>taskSwitchHookShow()</i> – show the list of task switch routines <i>taskDeleteHookShow()</i> – show the list of task delete routines |
| DESCRIPTION | <p>This library provides routines which summarize the installed kernel hook routines. There is one routine dedicated to the display of each type of kernel hook: task operation, task switch, and task deletion.</p> <p>The routine <i>taskHookShowInit()</i> links the task hook show facility into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none">– If you use configuration header files, define INCLUDE_SHOW_ROUTINES in config.h.– If you use the Tornado project facility, select INCLUDE_TASK_HOOK_SHOW. |
| INCLUDE FILES | taskHookLib.h |
| SEE ALSO | taskHookLib , <i>VxWorks Programmer's Guide: Basic OS</i> |

taskInfo

| | |
|--------------------|--|
| NAME | taskInfo – task information library |
| ROUTINES | <i>taskOptionsSet()</i> – change task options <i>taskOptionsGet()</i> – examine task options <i>taskRegsGet()</i> – get a task's registers from the TCB <i>taskRegsSet()</i> – set a task's registers <i>taskName()</i> – get the name associated with a task ID <i>taskNameToId()</i> – look up the task ID associated with a task name <i>taskIdDefault()</i> – set the default task ID <i>taskIsReady()</i> – check if a task is ready to run <i>taskIsSuspended()</i> – check if a task is suspended <i>taskIdListGet()</i> – get a list of active task IDs |
| DESCRIPTION | <p>This library provides a programmatic interface for obtaining task information.</p> <p>Task information is crucial as a debugging aid and user-interface convenience during the development cycle of an application. The routines <i>taskOptionsGet()</i>, <i>taskRegsGet()</i>,</p> |

taskName(), *taskNameToId()*, *taskIsReady()*, *taskIsSuspended()*, and *taskIdListGet()* are used to obtain task information. Three routines -- *taskOptionsSet()*, *taskRegsSet()*, and *taskIdDefault()* -- provide programmatic access to debugging features.

The chief drawback of using task information is that tasks may change their state between the time the information is gathered and the time it is utilized. Information provided by these routines should therefore be viewed as a snapshot of the system, and not relied upon unless the task is consigned to a known state, such as suspended.

Task management and control routines are provided by **taskLib**. Higher-level task information display routines are provided by *taskShow*.

INCLUDE FILES **taskLib.h**

SEE ALSO **taskLib**, **taskShow**, **taskHookLib**, **taskVarLib**, **semLib**, **kernelLib**, *VxWorks Programmer's Guide: Basic OS*

taskLib

NAME **taskLib** – task management library

ROUTINES *taskSpawn()* – spawn a task
taskInit() – initialize a task with a stack at a specified address
taskActivate() – activate a task that has been initialized
exit() – exit a task (ANSI)
taskDelete() – delete a task
taskDeleteForce() – delete a task without restriction
taskSuspend() – suspend a task
taskResume() – resume a task
taskRestart() – restart a task
taskPrioritySet() – change the priority of a task
taskPriorityGet() – examine the priority of a task
taskLock() – disable task rescheduling
taskUnlock() – enable task rescheduling
taskSafe() – make the calling task safe from deletion
taskUnsafe() – make the calling task unsafe from deletion
taskDelay() – delay a task from executing
taskIdSelf() – get the task ID of a running task
taskIdVerify() – verify the existence of a task
taskTcb() – get the task control block for a task ID

DESCRIPTION This library provides the interface to the VxWorks task management facilities. Task control services are provided by the VxWorks kernel, which is comprised of **kernelLib**, **taskLib**, **semLib**, **tickLib**, **msgQLib**, and **wdLib**. Programmatic access to task

information and debugging features is provided by **taskInfo**. Higher-level task information display routines are provided by **taskShow**.

TASK CREATION Tasks are created with the general-purpose routine *taskSpawn()*. Task creation consists of the following: allocation of memory for the stack and task control block (**WIND_TCB**), initialization of the **WIND_TCB**, and activation of the **WIND_TCB**. Special needs may require the use of the lower-level routines *taskInit()* and *taskActivate()*, which are the underlying primitives of *taskSpawn()*.

Tasks in VxWorks execute in the most privileged state of the underlying architecture. In a shared address space, processor privilege offers no protection advantages and actually hinders performance.

There is no limit to the number of tasks created in VxWorks, as long as sufficient memory is available to satisfy allocation requirements.

The routine *sp()* is provided in **usrLib** as a convenient abbreviation for spawning tasks. It calls *taskSpawn()* with default parameters.

TASK DELETION If a task exits its "main" routine, specified during task creation, the kernel implicitly calls *exit()* to delete the task. Tasks can be deleted with the *taskDelete()* or *exit()* routine.

Task deletion must be handled with extreme care, due to the inherent difficulties of resource reclamation. Deleting a task that owns a critical resource can cripple the system, since the resource may no longer be available. Simply returning a resource to an available state is not a viable solution, since the system can make no assumption as to the state of a particular resource at the time a task is deleted.

The solution to the task deletion problem lies in deletion protection, rather than overly complex deletion facilities. Tasks may be protected from unexpected deletion using *taskSafe()* and *taskUnsafe()*. While a task is safe from deletion, deleters will block until it is safe to proceed. Also, a task can protect itself from deletion by taking a mutual-exclusion semaphore created with the **SEM_DELETE_SAFE** option, which enables an implicit *taskSafe()* with each *semTake()*, and a *taskUnsafe()* with each *semGive()* (see **semMLib** for more information). Many VxWorks system resources are protected in this manner, and application designers may wish to consider this facility where dynamic task deletion is a possibility.

The **sigLib** facility may also be used to allow a task to execute clean-up code before actually expiring.

TASK CONTROL Tasks are manipulated by means of an ID that is returned when a task is created. VxWorks uses the convention that specifying a task ID of **NULL** in a task control function signifies the calling task.

The following routines control task state: *taskResume()*, *taskSuspend()*, *taskDelay()*, *taskRestart()*, *taskPrioritySet()*, and *taskRegsSet()*.

TASK SCHEDULING VxWorks schedules tasks on the basis of priority. Tasks may have priorities ranging from 0, the highest priority, to 255, the lowest priority. The priority of a task in VxWorks is dynamic, and an existing task's priority can be changed using *taskPrioritySet()*.

INCLUDE FILES **taskLib.h**

SEE ALSO **taskInfo**, **taskShow**, **taskHookLib**, **taskVarLib**, **semLib**, **semMLib**, **kernelLib**, *VxWorks Programmer's Guide: Basic OS*

taskShow

NAME **taskShow** – task show routines

ROUTINES *taskShowInit()* – initialize the task show routine facility
taskInfoGet() – get information about a task
taskShow() – display task information from TCBS
taskRegsShow() – display the contents of a task's registers
taskStatusString() – get a task's status as a string

DESCRIPTION This library provides routines to show task-related information, such as register values, task status, etc.

The *taskShowInit()* routine links the task show facility into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:

- If you use configuration header files, define **INCLUDE_SHOW_ROUTINES** in **config.h**.
- If you use the Tornado project facility, select **INCLUDE_TASK_SHOW**.

Task information is crucial as a debugging aid and user-interface convenience during the development cycle of an application. The routines *taskInfoGet()*, *taskShow()*, *taskRegsShow()*, and *taskStatusString()* are used to display task information.

The chief drawback of using task information is that tasks may change their state between the time the information is gathered and the time it is utilized. Information provided by these routines should therefore be viewed as a snapshot of the system, and not relied upon unless the task is consigned to a known state, such as suspended.

Task management and control routines are provided by **taskLib**. Programmatic access to task information and debugging features is provided by **taskInfo**.

INCLUDE FILES **taskLib.h**

SEE ALSO **taskLib**, **taskInfo**, **taskHookLib**, **taskVarLib**, **semLib**, **kernelLib**, *VxWorks Programmer's Guide: Basic OS*, *Target Shell*, *Tornado User's Guide: Shell*

taskVarLib

| | |
|----------------------|--|
| NAME | taskVarLib – task variables support library |
| ROUTINES | <i>taskVarInit()</i> – initialize the task variables facility <i>taskVarAdd()</i> – add a task variable to a task <i>taskVarDelete()</i> – remove a task variable from a task <i>taskVarGet()</i> – get the value of a task variable <i>taskVarSet()</i> – set the value of a task variable <i>taskVarInfo()</i> – get a list of task variables of a task |
| DESCRIPTION | <p>VxWorks provides a facility called "task variables," which allows 4-byte variables to be added to a task's context, and the variables' values to be switched each time a task switch occurs to or from the calling task. Typically, several tasks declare the same variable (4-byte memory location) as a task variable and treat that memory location as their own private variable. For example, this facility can be used when a routine must be spawned more than once as several simultaneous tasks.</p> <p>The routines <i>taskVarAdd()</i> and <i>taskVarDelete()</i> are used to add or delete a task variable. The routines <i>taskVarGet()</i> and <i>taskVarSet()</i> are used to get or set the value of a task variable.</p> |
| NOTE | If you are using task variables in a task delete hook (see taskHookLib), refer to the manual entry for <i>taskVarInit()</i> for warnings on proper usage. |
| INCLUDE FILES | taskVarLib.h |
| SEE ALSO | taskHookLib , <i>VxWorks Programmer's Guide: Basic OS</i> |

tcic

| | |
|--------------------|---|
| NAME | tcic – Databook TCIC/2 PCMCIA host bus adaptor chip driver |
| ROUTINES | <i>tcicInit()</i> – initialize the TCIC chip |
| DESCRIPTION | This library contains routines to manipulate the PCMCIA functions on the Databook DB86082 PCMCIA chip. The initialization routine <i>tcicInit()</i> is the only global function and is included in the PCMCIA chip table pcmciaAdapter . If <i>tcicInit()</i> finds the TCIC chip, it registers all function pointers of the PCMCIA_CHIP structure. |

tcicShow

| | |
|--------------------|---|
| NAME | tcicShow – Databook TCIC/2 PCMCIA host bus adaptor chip show library |
| ROUTINES | <i>tcicShow()</i> – show all configurations of the TCIC chip |
| DESCRIPTION | This is a driver show routine for the Databook DB86082 PCMCIA chip. <i>tcicShow()</i> is the only global function and is installed in the PCMCIA chip table pcmciaAdapter in <i>pcmciaShowInit()</i> . |

tcpShow

| | |
|--------------------|--|
| NAME | tcpShow – TCP information display routines |
| ROUTINES | <i>tcpShowInit()</i> – initialize TCP show routines <i>tcpDebugShow()</i> – display debugging information for the TCP protocol <i>tcpstatShow()</i> – display all statistics for the TCP protocol |
| DESCRIPTION | <p>This library provides routines to show TCP related statistics.</p> <p>Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:</p> <ul style="list-style-type: none">– <i>TCP/IP Illustrated Volume II, The Implementation</i>, by Richard Stevens– <i>The Design and Implementation of the 4.4 BSD UNIX Operating System</i>, by Leffler, McKusick, Karels and Quarterman <p>The <i>tcpShowInit()</i> routine links the TCP show facility into the VxWorks system. This is performed automatically if INCLUDE_NET_SHOW is defined in configAll.h.</p> |
| SEE ALSO | netLib , netShow , <i>VxWorks Programmer's Guide: Network</i> |

telnetLib

| | |
|----------------------|---|
| NAME | telnetLib – telnet server library |
| ROUTINES | <i>telnetInit()</i> – initialize the telnet daemon <i>telnetd()</i> – VxWorks telnet daemon |
| DESCRIPTION | <p>This library provides a remote login facility for VxWorks. It uses the telnet protocol to enable users on remote systems to log in to VxWorks.</p> <p>The telnet daemon, <i>telnetd()</i>, accepts remote telnet login requests and causes the shell's input and output to be redirected to the remote user. The telnet daemon is started by calling <i>telnetInit()</i>, which is called automatically when the configuration macro <code>INCLUDE_TELNET</code> is defined.</p> <p>Internally, the telnet daemon provides a tty-like interface to the remote user through the use of the VxWorks pseudo-terminal driver, <code>ptyDrv</code>.</p> |
| INCLUDE FILES | telnetLib.h |
| SEE ALSO | ptyDrv, rlogLib |

tftpdLib

| | |
|--------------------|--|
| NAME | tftpdLib – Trivial File Transfer Protocol server library |
| ROUTINES | <i>tftpdInit()</i> – initialize the TFTP server task <i>tftpdTask()</i> – TFTP server daemon task <i>tftpdDirectoryAdd()</i> – add a directory to the access list <i>tftpdDirectoryRemove()</i> – delete a directory from the access list |
| DESCRIPTION | <p>This library implements the VxWorks Trivial File Transfer Protocol (TFTP) server module. The server can respond to both read and write requests. It is started by a call to <i>tftpdInit()</i>.</p> <p>The server has access to a list of directories that can either be provided in the initial call to <i>tftpdInit()</i> or changed dynamically using the <i>tftpdDirectoryAdd()</i> and <i>tftpdDirectoryRemove()</i> calls. Requests for files not in the directory trees specified in the access list will be rejected, unless the list is empty, in which case all requests will be allowed. By default, the access list contains the directory given in the global variable <code>tftpdDirectory</code>. It is possible to remove the default by calling <i>tftpdDirectoryRemove()</i>.</p> |

For specific information about the TFTP protocol, see RFC 783, "TFTP Protocol."

INCLUDE FILES `ftplib.h, tftplib.h`

SEE ALSO `ftplib`, RFC 783 "TFTP Protocol", *VxWorks Programmer's Guide: Network*

ftplib

NAME `ftplib` – Trivial File Transfer Protocol (TFTP) client library

ROUTINES `tftpXfer()` – transfer a file via TFTP using a stream interface
`tftpCopy()` – transfer a file via TFTP
`tftpInit()` – initialize a TFTP session
`tftpModeSet()` – set the TFTP transfer mode
`tftpPeerSet()` – set the TFTP server address
`tftpPut()` – put a file to a remote system
`tftpGet()` – get a file from a remote system
`tftpInfoShow()` – get TFTP status information
`tftpQuit()` – quit a TFTP session
`tftpSend()` – send a TFTP message to the remote system

DESCRIPTION This library implements the VxWorks Trivial File Transfer Protocol (TFTP) client library. TFTP is a simple file transfer protocol (hence the name "trivial") implemented over UDP. TFTP was designed to be small and easy to implement; therefore it is limited in functionality in comparison with other file transfer protocols, such as FTP. TFTP provides only the read/write capability to and from a remote server.

TFTP provides no user authentication; therefore the remote files must have "loose" permissions before requests for file access will be granted by the remote TFTP server (i.e., files to be read must be publicly readable, and files to be written must exist and be publicly writeable). Some TFTP servers offer a secure option (-s) that specifies a directory where the TFTP server is rooted. Refer to the host manuals for more information about a particular TFTP server.

HIGH-LEVEL INTERFACE

The `ftplib` library has two levels of interface. The tasks `tftpXfer()` and `tftpCopy()` operate at the highest level and are the main call interfaces. The `tftpXfer()` routine provides a stream interface to TFTP. That is, it spawns a task to perform the TFTP transfer and provides a descriptor from which data can be transferred interactively. The `tftpXfer()` interface is similar to `ftpXfer()` in `ftplib`. The `tftpCopy()` routine transfers a remote file to or from a passed file (descriptor).

LOW-LEVEL INTERFACE

The lower-level interface is made up of various routines that act on a TFTP session. Each TFTP session is defined by a TFTP descriptor. These routines include:

- tftpInit()* to initialize a session;
- tftpModeSet()* to set the transfer mode;
- tftpPeerSet()* to set a peer/server address;
- tftpPut()* to put a file to the remote system;
- tftpGet()* to get file from remote system;
- tftpInfoShow()* to show status information; and
- tftpQuit()* to quit a TFTP session.

EXAMPLE

The following code provides an example of how to use the lower-level routines. It implements roughly the same function as *tftpCopy()*.

```
char *      pHost;
int         port;
char *      pFilename;
char *      pCommand;
char *      pMode;
int         fd;
TFTP_DESC * pTftpDesc;
int         status;
if ((pTftpDesc = tftpInit ()) == NULL)
    return (ERROR);
if ((tftpPeerSet (pTftpDesc, pHost, port) == ERROR) ||
    (tftpModeSet (pTftpDesc, pMode) == ERROR))
    {
        (void) tftpQuit (pTftpDesc);
        return (ERROR);
    }
if (strcmp (pCommand, "get") == 0)
    {
        status = tftpGet (pTftpDesc, pFilename, fd, TFTP_CLIENT);
    }
else if (strcmp (pCommand, "put") == 0)
    {
        status = tftpPut (pTftpDesc, pFilename, fd, TFTP_CLIENT);
    }
else
    {
        errno = S_tftpLib_INVALID_COMMAND;
        status = ERROR;
    }
(void) tftpQuit (pTftpDesc);
```

INCLUDE FILES **tftpLib.h**

SEE ALSO **tftpLib**, *VxWorks Programmer's Guide: Network*

tickLib

NAME **tickLib** – clock tick support library

ROUTINES *tickAnnounce()* – announce a clock tick to the kernel
tickSet() – set the value of the kernel's tick counter
tickGet() – get the value of the kernel's tick counter

DESCRIPTION This library is the interface to the VxWorks kernel routines that announce a clock tick to the kernel, get the current time in ticks, and set the current time in ticks.

Kernel facilities that rely on clock ticks include *taskDelay()*, *wdStart()*, *kernelTimeslice()*, and semaphore timeouts. In each case, the specified timeout is relative to the current time, also referred to as "time to fire." Relative timeouts are not affected by calls to *tickSet()*, which only changes absolute time. The routines *tickSet()* and *tickGet()* keep track of absolute time in isolation from the rest of the kernel.

Time-of-day clocks or other auxiliary time bases are preferable for lengthy timeouts of days or more. The accuracy of such time bases is greater, and some external time bases even calibrate themselves periodically.

INCLUDE FILES **tickLib.h**

SEE ALSO **kernelLib**, **taskLib**, **semLib**, **wdLib**, *VxWorks Programmer's Guide: Basic OS*

timerLib

NAME **timerLib** – timer library (POSIX)

ROUTINES *timer_cancel()* – cancel a timer
timer_connect() – connect a user routine to the timer signal
timer_create() – allocate a timer using the specified clock for a timing base (POSIX)
timer_delete() – remove a previously created timer (POSIX)
timer_gettime() – get the remaining time before expiration and the reload value (POSIX)
timer_getoverrun() – return the timer expiration overrun (POSIX)

timexLib

timer_settime() – set the time until the next expiration and arm timer (POSIX)

nanosleep() – suspend the current task until the time interval elapses (POSIX)

| | |
|-----------------------|---|
| DESCRIPTION | <p>This library provides a timer interface, as defined in the IEEE standard, POSIX 1003.1b.</p> <p>Timers are mechanisms by which tasks signal themselves after a designated interval. Timers are built on top of the clock and signal facilities. The clock facility provides an absolute time-base. Standard timer functions simply consist of creation, deletion and setting of a timer. When a timer expires, <i>sigaction()</i> (see sigLib) must be in place in order for the user to handle the event. The "high resolution sleep" facility, <i>nanosleep()</i>, allows sub-second sleeping to the resolution of the clock.</p> <p>The clockLib library should be installed and <i>clock_settime()</i> set before the use of any timer routines.</p> |
| ADDITIONS | <p>Two non-POSIX functions are provided for user convenience:</p> <p><i>timer_cancel()</i> quickly disables a timer by calling <i>timer_settime()</i>.</p> <p><i>timer_connect()</i> easily hooks up a user routine by calling <i>sigaction()</i>.</p> |
| CLARIFICATIONS | <p>The task creating a timer with <i>timer_create()</i> will receive the signal no matter which task actually arms the timer.</p> <p>When a timer expires and the task has previously exited, <i>logMsg()</i> indicates the expected task is not present. Similarly, <i>logMsg()</i> indicates when a task arms a timer without installing a signal handler. Timers may be armed but not created or deleted at interrupt level.</p> |
| IMPLEMENTATION | <p>The actual clock resolution is hardware-specific and in many cases is 1/60th of a second. This is less than <code>_POSIX_CLOCKRES_MIN</code>, which is defined as 20 milliseconds (1/50th of a second).</p> |
| INCLUDE FILES | timers.h |
| SEE ALSO | clockLib , <i>sigaction()</i> , POSIX 1003.1b documentation, <i>VxWorks Programmer's Guide: Basic OS</i> |

timexLib

| | |
|-----------------|--|
| NAME | timexLib – execution timer facilities |
| ROUTINES | <p><i>timexInit()</i> – include the execution timer library</p> <p><i>timexClear()</i> – clear the list of function calls to be timed</p> <p><i>timexFunc()</i> – specify functions to be timed</p> <p><i>timexHelp()</i> – display synopsis of execution timer facilities</p> |

timex() – time a single execution of a function or functions
timexN() – time repeated executions of a function or group of functions
timexPost() – specify functions to be called after timing
timexPre() – specify functions to be called prior to timing
timexShow() – display the list of function calls to be timed

DESCRIPTION

This library contains routines for timing the execution of programs, individual functions, and groups of functions. The VxWorks system clock is used as a time base. Functions that have a short execution time relative to this time base can be called repeatedly to establish an average execution time with an acceptable percentage of error.

Up to four functions can be specified to be timed as a group. Additionally, sets of up to four functions can be specified as pre- or post-timing functions, to be executed before and after the timed functions. The routines *timexPre()* and *timexPost()* are used to specify the pre- and post-timing functions, while *timexFunc()* specifies the functions to be timed.

The routine *timex()* is used to time a single execution of a function or group of functions. If called with no arguments, *timex()* uses the functions in the lists created by calls to *timexPre()*, *timexPost()*, and *timexFunc()*. If called with arguments, *timex()* times the function specified, instead of the previous list. The routine *timexN()* works in the same manner as *timex()* except that it iterates the function calls to be timed.

EXAMPLES

The routine *timex()* can be used to obtain the execution time of a single routine:

```
-> timex myFunc, myArg1, myArg2, ...
```

The routine *timexN()* calls a function repeatedly until a 2% or better tolerance is obtained:

```
-> timexN myFunc, myArg1, myArg2, ...
```

The routines *timexPre()*, *timexPost()*, and *timexFunc()* are used to specify a list of functions to be executed as a group:

```
-> timexPre 0, myPreFunc1, preArg1, preArg2, ...
-> timexPre 1, myPreFunc2, preArg1, preArg2, ...
-> timexFunc 0, myFunc1, myArg1, myArg2, ...
-> timexFunc 1, myFunc2, myArg1, myArg2, ...
-> timexFunc 2, myFunc3, myArg1, myArg2, ...
-> timexPost 0, myPostFunc, postArg1, postArg2, ...
```

The list is executed by calling *timex()* or *timexN()* without arguments:

```
-> timex
```

or

```
-> timexN
```

In this example, *myPreFunc1* and *myPreFunc2* are called with their respective arguments. *myFunc1*, *myFunc2*, and *myFunc3* are then called in sequence and timed. If *timexN()* was used, the sequence is called repeatedly until a 2% or better error tolerance is achieved.

Finally, *myPostFunc* is called with its arguments. The timing results are reported after all post-timing functions are called.

NOTE The timings measure the execution time of the routine body, without the usual subroutine entry and exit code (usually LINK, UNLINK, and RTS instructions). Also, the time required to set up the arguments and call the routines is not included in the reported times. This is because these timing routines automatically calibrate themselves by timing the invocation of a null routine, and thereafter subtracting that constant overhead.

INCLUDE FILES **timexLib.h**

SEE ALSO **spyLib**

ttyDrv

NAME **ttyDrv** – provide terminal device access to serial channels

ROUTINES *ttyDrv()* – initialize the tty driver
ttyDevCreate() – create a VxWorks device for a serial channel

DESCRIPTION This library provides the OS-dependent functionality of a serial device, including canonical processing and the interface to the VxWorks I/O system.

The BSP provides "raw" serial channels which are accessed via an **SIO_CHAN** data structure. These raw devices provide only low level access to the devices to send and receive characters. This library builds on that functionality by allowing the serial channels to be accessed via the VxWorks I/O system using the standard read/write interface. It also provides the canonical processing support of **tyLib**.

The routines in this library are typically called by *usrRoot()* in **usrConfig.c** to create VxWorks serial devices at system startup time.

INCLUDE FILES **ttyLib.h**

SEE ALSO **tyLib, sioLib.h**

tyLib

| | |
|--------------------|---|
| NAME | tyLib – tty driver support library |
| ROUTINES | <i>tyDevInit()</i> – initialize the tty device descriptor <i>tyAbortFuncSet()</i> – set the abort function <i>tyAbortSet()</i> – change the abort character <i>tyBackspaceSet()</i> – change the backspace character <i>tyDeleteLineSet()</i> – change the line-delete character <i>tyEOFSet()</i> – change the end-of-file character <i>tyMonitorTrapSet()</i> – change the trap-to-monitor character <i>tyIoctl()</i> – handle device control requests <i>tyWrite()</i> – do a task-level write for a tty device <i>tyRead()</i> – do a task-level read for a tty device <i>tyITx()</i> – interrupt-level output <i>tyIRd()</i> – interrupt-level input |
| DESCRIPTION | <p>This library provides routines used to implement drivers for serial devices. It provides all the necessary device-independent functions of a normal serial channel, including:</p> <ul style="list-style-type: none">– ring buffering of input and output– raw mode– optional line mode with backspace and line-delete functions– optional processing of X-on/X-off– optional RETURN/LINEFEED conversion– optional echoing of input characters– optional stripping of the parity bit from 8-bit input– optional special characters for shell abort and system restart <p>Most of the routines in this library are called only by device drivers. Functions that normally might be called by an application or interactive user are the routines to set special characters, <i>ty...Set()</i>.</p> |

USE IN SERIAL DEVICE DRIVERS

Each device that uses **tyLib** is described by a data structure of type **TY_DEV**. This structure begins with an I/O system device header so that it can be added directly to the I/O system's device list. A driver calls *tyDevInit()* to initialize a **TY_DEV** structure for a specific device and then calls *iosDevAdd()* to add the device to the I/O system.

The call to *tyDevInit()* takes three parameters: the pointer to the **TY_DEV** structure to initialize, the desired size of the read and write ring buffers, and the address of a transmitter start-up routine. This routine will be called when characters are added for output and the transmitter is idle. Thereafter, the driver can call the following routines to perform the usual device functions:

tyLib

tyRead()

user read request to get characters that have been input

tyWrite()

user write request to put characters to be output

tyIoctl()

user I/O control request

tyIRd()

interrupt-level routine to get an input character

tyITx()

interrupt-level routine to deliver the next output character

Thus, *tyRead()*, *tyWrite()*, and *tyIoctl()* are called from the driver's read, write, and I/O control functions. The routines *tyIRd()* and *tyITx()* are called from the driver's interrupt handler in response to receive and transmit interrupts, respectively.

Examples of using **tyLib** in a driver can be found in the source file(s) included by tyCoDrv. Source files are located in src/drv/serial.

TTY OPTIONS

A full range of options affects the behavior of tty devices. These options are selected by setting bits in the device option word using the **FIOSETOPTIONS** function in the *ioctl()* routine (see "I/O Control Functions" below for more information). The following is a list of available options. The options are defined in the header file **ioLib.h**.

OPT_LINE

Selects line mode. A tty device operates in one of two modes: raw mode (unbuffered) or line mode. Raw mode is the default. In raw mode, each byte of input from the device is immediately available to readers, and the input is not modified except as directed by other options below. In line mode, input from the device is not available to readers until a NEWLINE character is received, and the input may be modified by backspace, line-delete, and end-of-file special characters.

OPT_ECHO

Causes all input characters to be echoed to the output of the same channel. This is done simply by putting incoming characters in the output ring as well as the input ring. If the output ring is full, the echoing is lost without affecting the input.

OPT_CRMOD

C language conventions use the NEWLINE character as the line terminator on both input and output. Most terminals, however, supply a RETURN character when the return key is hit, and require both a RETURN and a LINEFEED character to advance the output line. This option enables the appropriate translation: NEWLINES are substituted for input RETURN characters, and NEWLINES in the output file are automatically turned into a RETURN-LINEFEED sequence.

OPT_TANDEM

Causes the driver to generate and respond to the special flow control characters

CTRL-Q and CTRL-S in what is commonly known as X-on/X-off protocol. Receipt of a CTRL-S input character will suspend output to that channel. Subsequent receipt of a CTRL-Q will resume the output. Also, when the VxWorks input buffer is almost full, a CTRL-S will be output to signal the other side to suspend transmission. When the input buffer is almost empty, a CTRL-Q will be output to signal the other side to resume transmission.

OPT_7_BIT

Strips the most significant bit from all bytes input from the device.

OPT_MON_TRAP

Enables the special monitor trap character, by default CTRL-X. When this character is received and this option is enabled, VxWorks will trap to the ROM resident monitor program. Note that this is quite drastic. All normal VxWorks functioning is suspended, and the computer system is entirely controlled by the monitor. Depending on the particular monitor, it may or may not be possible to restart VxWorks from the point of interruption. The default monitor trap character can be changed by calling *tyMonitorTrapSet()*.

OPT_ABORT

Enables the special shell abort character, by default CTRL-C. When this character is received and this option is enabled, the VxWorks shell is restarted. This is useful for freeing a shell stuck in an unfriendly routine, such as one caught in an infinite loop or one that has taken an unavailable semaphore. For more information, see the *VxWorks Programmer's Guide: Shell*.

OPT_TERMINAL

This is not a separate option bit. It is the value of the option word with all the above bits set.

OPT_RAW

This is not a separate option bit. It is the value of the option word with none of the above bits set.

I/O CONTROL FUNCTIONS

The tty devices respond to the following *ioctl()* functions. The functions are defined in the header **ioLib.h**.

FIOGETNAME

Gets the file name of the file descriptor and copies it to the buffer referenced to by *nameBuf*:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

This function is common to all file descriptors for all devices.

FIOSETOPTIONS, FIOOPTIONS

Sets the device option word to the specified argument. For example, the call:

```
status = ioctl (fd, FIOOPTIONS, OPT_TERMINAL);  
status = ioctl (fd, FIOSETOPTIONS, OPT_TERMINAL);
```

enables all the tty options described above, putting the device in a "normal" terminal mode. If the line protocol (**OPT_LINE**) is changed, the input buffer is flushed. The various options are described in **ioLib.h**.

FIOGETOPTIONS

Returns the current device option word:

```
options = ioctl (fd, FIOGETOPTIONS, 0);
```

FIONREAD

Copies to *nBytesUnread* the number of bytes available to be read in the device's input buffer:

```
status = ioctl (fd, FIONREAD, &nBytesUnread);
```

In line mode (**OPT_LINE** set), the **FIONREAD** function actually returns the number of characters available plus the number of lines in the buffer. Thus, if five lines of just **NEWLINEs** were in the input buffer, it would return the value 10 (5 characters + 5 lines).

FIONWRITE

Copies to *nBytes* the number of bytes queued to be output in the device's output buffer:

```
status = ioctl (fd, FIONWRITE, &nBytes);
```

FIOFLUSH

Discards all the bytes currently in both the input and the output buffers:

```
status = ioctl (fd, FIOFLUSH, 0);
```

FIOWFLUSH

Discards all the bytes currently in the output buffer:

```
status = ioctl (fd, FIOWFLUSH, 0);
```

FIORFLUSH

Discards all the bytes currently in the input buffers:

```
status = ioctl (fd, FIORFLUSH, 0);
```

FIOCANCEL

Cancels a read or write. A task blocked on a read or write may be released by a second task using this *ioctl()* call. For example, a task doing a read can set a watchdog timer before attempting the read; the auxiliary task would wait on a semaphore. The watchdog routine can give the semaphore to the auxiliary task, which would then use the following call on the appropriate file descriptor:

```
status = ioctl (fd, FIOCANCEL, 0);
```

FIOBAUDRATE

Sets the baud rate of the device to the specified argument. For example, the call:

```
status = ioctl (fd, FIOBAUDRATE, 9600);
```

Sets the device to operate at 9600 baud. This request has no meaning on a pseudo terminal.

FIOISATTY

Returns TRUE for a tty device:

```
status = ioctl (fd, FIOISATTY, 0);
```

FIOPROTOHOOK

Adds a protocol hook function to be called for each input character. *pfunction* is a pointer to the protocol hook routine which takes two arguments of type *int* and returns values of type STATUS (TRUE or FALSE). The first argument passed is set by the user via the **FIOPROTOARG** function. The second argument is the input character. If no further processing of the character is required by the calling routine (the input routine of the driver), the protocol hook routine *pFunction* should return TRUE. Otherwise, it should return FALSE:

```
status = ioctl (fd, FIOPROTOHOOK, pFunction);
```

FIOPROTOARG

Sets the first argument to be passed to the protocol hook routine set by **FIOPROTOHOOK** function:

```
status = ioctl (fd, FIOPROTOARG, arg);
```

FIORBUFSET

Changes the size of the receive-side buffer to *size*:

```
status = ioctl (fd, FIORBUFSET, size);
```

FIOWBUFSET

Changes the size of the send-side buffer to *size*:

```
status = ioctl (fd, FIOWBUFSET, size);
```

Any other *ioctl()* request will return an error and set the status to **S_ioLib_UNKNOWN_REQUEST**.

INCLUDE FILES **tyLib.h, ioLib.h**

SEE ALSO **ioLib, iosLib, tyCoDrv, VxWorks Programmer's Guide: I/O System**

udpShow

NAME **udpShow** – UDP information display routines

ROUTINES **udpShowInit()** – initialize UDP show routines
udpstatShow() – display statistics for the UDP protocol

DESCRIPTION This library provides routines to show UDP related statistics.

Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:

- *TCP/IP Illustrated Volume II, The Implementation*, by Richard Stevens
- *The Design and Implementation of the 4.4 BSD UNIX Operating System*, by Leffler, McKusick, Karels and Quarterman

The `udpShowInit()` routine links the UDP show facility into the VxWorks system. This is performed automatically if `INCLUDE_NET_SHOW` is defined in `configAll.h`.

SEE ALSO `udpShow`, `netLib`, `netShow`, *VxWorks Programmer's Guide: Network*

ultraEnd

NAME `ultraEnd` – SMC Ultra Elite END network interface driver

ROUTINES `ultraLoad()` – initialize the driver and device
`ultraParse()` – parse the init string
`ultraMemInit()` – initialize memory for the chip
`ultraAddrFilterSet()` – set the address filter for multicast addresses

DESCRIPTION This module implements the SMC Elite Ultra Ethernet network interface driver.

This driver supports single transmission and multiple reception. The Current register is a write pointer to the ring. The Bound register is a read pointer from the ring. This driver gets the Current register at the interrupt level and sets the Bound register at the task level. The interrupt is only masked during configuration or in polled mode.

CONFIGURATION The W1 jumper should be set in the position of "Software Configuration". The defined I/O address in `config.h` must match the one stored in EEROM. The RAM address, the RAM size, and the IRQ level are defined in `config.h`. IRQ levels 2,3,5,7,10,11,15 are supported.

EXTERNAL SUPPORT REQUIREMENTS

This driver requires several external support functions, defined as macros:

```
SYS_INT_CONNECT(pDrvCtrl, routine, arg)
SYS_INT_DISCONNECT (pDrvCtrl, routine, arg)
SYS_INT_ENABLE(pDrvCtrl)
SYS_INT_DISABLE(pDrvCtrl)
SYS_IN_BYTE(pDrvCtrl, reg, pData)
SYS_OUT_BYTE(pDrvCtrl, reg, pData)
```

These macros allow the driver to be customized for BSPs that use special versions of these routines.

The macro **SYS_INT_CONNECT** is used to connect the interrupt handler to the appropriate vector. By default it is the routine *intConnect()*.

The macro **SYS_INT_DISCONNECT** is used to disconnect the interrupt handler prior to unloading the module. By default this is a dummy routine that returns OK.

The macro **SYS_INT_ENABLE** is used to enable the interrupt level for the end device. It is called once during initialization. It calls an external board level routine *sysUltraIntEnable()*.

The macro **SYS_INT_DISABLE** is used to disable the interrupt level for the end device. It is called once during shutdown. It calls an external board level routine *sysUltraIntDisable()*.

The macros **SYS_IN_BYTE** and **SYS_OUT_BYTE** are used for accessing the ultra device. The default macros map these operations onto *sysInByte()* and *sysOutByte()*.

INCLUDES **end.h endLib.h etherMultiLib.h**

SEE ALSO **ultraEnd, muxLib, endLib, *Writing an Enhanced Network Driver***

unixDrv

NAME **unixDrv** – UNIX-file disk driver (VxSim for Solaris and VxSim for HP)

ROUTINES *unixDrv()* – install UNIX disk driver
unixDiskDevCreate() – create a UNIX disk device
unixDiskInit() – initialize a dosFs disk on top of UNIX

DESCRIPTION This driver emulates a VxWorks disk driver, but actually uses the UNIX file system to store the data. The VxWorks disk appears under UNIX as a single file. The UNIX file name, and the size of the disk, may be specified during the *unixDiskDevCreate()* call.

USER-CALLABLE ROUTINES

Most of the routines in this driver are accessible only through the I/O system. The routine *unixDrv()* must be called to initialize the driver and the *unixDiskDevCreate()* routine is used to create devices.

CREATING UNIX DISKS

Before a UNIX disk can be used, it must be created. This is done with the *unixDiskDevCreate()* call. The format of this call is:

```
BLK_DEV *unixDiskDevCreate
(
    char    *unixFile,      /* name of the UNIX file to use      */
    int     bytesPerBlk,    /* number of bytes per block         */
    int     blksPerTrack,   /* number of blocks per track        */
    int     nBlocks        /* number of blocks on this device    */
)
```

The UNIX file must be pre-allocated separately. This can be done using the UNIX `mkfile(8)` command. Note that you have to create an appropriately sized file. For example, to create a UNIX file system that is used as a common floppy dosFs file system, you would issue the command:

```
mkfile 1440k /tmp/floppy.dos
```

This will create space for a 1.44 Meg DOS floppy (1474560 bytes, or 2880 512-byte blocks).

The *bytesPerBlk* parameter specifies the size of each logical block on the disk. If *bytesPerBlk* is zero, 512 is the default.

The *blksPerTrack* parameter specifies the number of blocks on each logical track of the UNIX disk. If *blksPerTrack* is zero, the count of blocks per track will be set to *nBlocks* (i.e., the disk will be defined as having only one track). UNIX disk devices typically are specified with only one track.

The *nBlocks* parameter specifies the size of the disk, in blocks. If *nBlocks* is zero the size of the UNIX file specified, divided by the number of bytes per block, is used.

The formatting parameters (*bytesPerBlk*, *blksPerTrack*, and *nBlocks*) are critical only if the UNIX disk already contains the contents of a disk created elsewhere. In that case, the formatting parameters must be identical to those used when the image was created. Otherwise, they may be any convenient number.

Once the device has been created it still does not have a name or file system associated with it. This must be done by using the file system's device initialization routine (e.g., *dosFsDevInit()*). The dosFs and rt11Fs file systems also provide make-file-system routines (*dosFsMkfs()* and *rt11FsMkfs()*), which may be used to associate a name and file system with the block device and initialize that file system on the device using default configuration parameters.

The *unixDiskDevCreate()* call returns a pointer to a block device structure (**BLK_DEV**). This structure contains fields that describe the physical properties of a disk device and specify the addresses of routines within the UNIX disk driver. The **BLK_DEV** structure address must be passed to the desired file system (dosFs, rt11Fs, or rawFs) during the file system's device initialization or make-file-system routine. Only then is a name and file system associated with the device, making it available for use.

As an example, to create a 200KB disk, 512-byte blocks, and only one track, the proper call would be:

```
BLK_DEV *pBlkDev;
pBlkDev = unixDiskDevCreate ("/tmp/filesys1", 512, 400, 400, 0);
```

This will attach the UNIX file /tmp/filesys1 as a block device.

A convenience routine, *unixDiskInit()*, is provided to do the *unixDiskDevCreate()* followed by either a *dosFsMkFs()* or *dosFsDevInit()*, whichever is appropriate.

The format of this call is:

```
BLK_DEV *unixDiskInit
(
    char * unixFile, /* name of the UNIX file to use */
    char * volName, /* name of the dosFs volume to use */
    int   nBytes    /* number of bytes in dosFs volume */
)
```

This call will create the UNIX disk if required.

- IOCTL** Only the FIODISKFORMAT request is supported; all other ioctl requests return an error, and set the task's errno to S_ioLib_UNKNOWN_REQUEST.
- SEE ALSO** *unixDrv*, *dosFsDevInit()*, *dosFsMkfs()*, *rt11FsDevInit()*, *rt11FsMkfs()*, *rawFsDevInit()*, *VxWorks Programmer's Guide: I/O System, Local File Systems*

unixSio

- NAME** *unixSio* – unix serial driver
- ROUTINES** *unixDevInit()* – initialize a UNIX_DUSART
unixDevInit2() – enable interrupts
unixIntRcv() – handle a channel's receive-character interrupt.
dummyCallback() – dummy callback routine.
- DESCRIPTION** This is the driver for the UNIX stdin/stdio-base simulated serial port.
- USAGE** A UNIX_CHAN structure is used to describe each channel available.
- The BSP's *sysHwInit()* routine typically calls *sysSerial.c:sysSerialHwInit()*, which opens UNIX tty/pty devices for serial lines and initializes the UNIX_CHAN *u_fd* and *u_pid* fields before calling *unixDevInit()*.
- The BSP *sysSerialHwInit2()* calls *unixDevInit2()* to enable interrupts.
- ```
#include "drv/sio/unixSio.h"
UNIX_CHAN myChan [NUM_TTY];
```

**unldLib**

```

SIO_CHAN * sysSioChans[NUM_TTY];
sysSerialHwInit (void)
{
 ...
 for (ix = 0; ix < NUM_TTY; ix++)
 {
 if (ix > 0) // dev 0 is unix sdtin/out/err //
 {
 UNIX_CHAN * pChan = &myChan[ix];
 sysSioChans[ix] = (SIO_CHAN *) pChan;
 pChan->u_fd = ptyXtermOpen (ptyName, &pChan->u_pid, 0);
 }
 ...
 unixDevInit (&myChan);
 }
}
sysSerialHwInit2 (void)
{
 ...
 for (i = 0; i < NUM_TTY; i++)
 intConnect (FD_TO_IVEC(myChan[i]->u_fd), unixInt, (int)&myChan[i]);
 ...
}

```

INCLUDE FILES    drv/sio/unixSio.h sioLib.h

---

## unldLib

**NAME**            **unldLib** – object module unloading library

**ROUTINES**        *unld()* – unload an object module by specifying a file name or module ID  
*unldByModuleId()* – unload an object module by specifying a module ID  
*unldByNameAndPath()* – unload an object module by specifying a name and path  
*unldByGroup()* – unload an object module by specifying a group number  
*reld()* – reload an object module

**DESCRIPTION**    This library provides a facility for unloading object modules. Once an object module has been loaded into the system (using the facilities provided by **loadLib**), it can be removed from the system by calling one of the *unld...()* routines in this library.

Unloading of an object module does the following:

- (1) It frees the space allocated for text, data, and BSS segments, unless *loadModuleAt()*



was called with specific addresses, in which case the user is responsible for freeing the space.

- (2) It removes all symbols associated with the object module from the system symbol table.
- (3) It removes the module descriptor from the module list.

Once the module is unloaded, any calls to routines in that module from other modules will fail unpredictably. The user is responsible for ensuring that no modules are unloaded that are used by other modules. *unld()* checks the hooks created by the following routines to ensure none of the unloaded code is in use by a hook:

```
taskCreateHookAdd()
taskDeleteHookAdd()
taskHookAdd()
taskSwapHookAdd()
taskSwitchHookAdd()
```

However, *unld()* *does not* check the hooks created by these routines:

```
etherInputHookAdd()
etherOutputHookAdd()
excHookAdd()
rebootHookAdd()
moduleCreateHookAdd()
```

**INCLUDE FILES**     **unldLib.h, moduleLib.h**

**SEE ALSO**         **loadLib, moduleLib**

---

## usrAta

**NAME**             **usrAta** – ATA initialization

**ROUTINES**        *usrAtaConfig()* – mount a DOS file system from an ATA hard disk  
*usrAtaPartition()* – get an offset to the first partition of the drive

## usrConfig

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>usrConfig</b> – user-defined system configuration library                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>ROUTINES</b>      | <i>usrInit()</i> – user-defined system initialization routine<br><i>usrRoot()</i> – the root task<br><i>usrClock()</i> – user-defined system clock interrupt routine                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>DESCRIPTION</b>   | <p>This library is the WRS-supplied configuration module for VxWorks. It contains the root task, the primary system initialization routine, the network initialization routine, and the clock interrupt routine.</p> <p>The include file <b>config.h</b> includes a number of system-dependent parameters used in this file.</p> <p>In an effort to simplify the presentation of the configuration of vxWorks, this file has been split into smaller files. These additional configuration source files are located in <code>../src/config/usr[xxx].c</code> and are <code>#included</code> into this file below. This file contains the bulk of the code a customer is likely to customize.</p> <p>The module <b>usrDepend.c</b> contains checks that guard against unsupported configurations such as <code>INCLUDE_NFS</code> without <code>INCLUDE_RPC</code>. The module <b>usrKernel.c</b> contains the core initialization of the kernel which is rarely customized, but provided for information. The module <b>usrNetwork.c</b> now contains all network initialization code. Finally, the module <b>usrExtra.c</b> contains the conditional inclusion of the optional packages selected in <b>configAll.h</b>.</p> <p>The source code necessary for the configuration selected is entirely included in this file during compilation as part of a standard build in the board support package. No other make is necessary.</p> |
| <b>INCLUDE FILES</b> | <b>config.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>      | <i>VxWorks Programmer's Guide: Configuration &amp; Build</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

---

## usrFd

|                 |                                                                   |
|-----------------|-------------------------------------------------------------------|
| <b>NAME</b>     | <b>usrFd</b> – floppy disk initialization                         |
| <b>ROUTINES</b> | <i>usrFdConfig()</i> – mount a DOS file system from a floppy disk |

---

## usrIde

**NAME** **usrIde** – IDE initialization

**ROUTINES** *usrIdeConfig()* – mount a DOS file system from an IDE hard disk

---

## usrLib

**NAME** **usrLib** – user interface subroutine library

**ROUTINES** *help()* – print a synopsis of selected routines  
*netHelp()* – print a synopsis of network routines  
*bootChange()* – change the boot line  
*periodRun()* – call a function periodically  
*period()* – spawn a task to call a function periodically  
*repeatRun()* – call a function repeatedly  
*repeat()* – spawn a task to call a function repeatedly  
*sp()* – spawn a task with default parameters  
*checkStack()* – print a summary of each task's stack usage  
*i()* – print a summary of each task's TCB  
*ti()* – print complete information from a task's TCB  
*show()* – print information on a specified object  
*ts()* – suspend a task  
*tr()* – resume a task  
*td()* – delete a task  
*version()* – print VxWorks version information  
*m()* – modify memory  
*d()* – display memory  
*cd()* – change the default directory  
*pwd()* – print the current default directory  
*copy()* – copy *in* (or stdin) to *out* (or stdout)  
*copyStreams()* – copy from/to specified streams  
*diskFormat()* – format a disk  
*diskInit()* – initialize a file system on a block device  
*squeeze()* – reclaim fragmented free space on an RT-11 volume  
*ld()* – load an object module into memory  
*ls()* – list the contents of a directory  
*ll()* – do a long listing of directory contents  
*lsOld()* – list the contents of an RT-11 directory  
*mkdir()* – make a directory

**usrLib**

*rmdir()* – remove a directory  
*rm()* – remove a file  
*devs()* – list all system-known devices  
*lkup()* – list symbols  
*lkAddr()* – list symbols whose values are near a specified value  
*mRegs()* – modify registers  
*pc()* – return the contents of the program counter  
*printErrno()* – print the definition of a specified error status value  
*printLogo()* – print the VxWorks logo  
*logout()* – log out of the VxWorks system  
*h()* – display or set the size of shell history  
*spyReport()* – display task activity data  
*spyTask()* – run periodic task activity reports  
*spy()* – begin periodic task activity reports  
*spyClkStart()* – start collecting task activity data  
*spyClkStop()* – stop collecting task activity data  
*spyStop()* – stop spying and reporting  
*spyHelp()* – display task monitoring help menu

**DESCRIPTION** This library consists of routines meant to be executed from the VxWorks shell. It provides useful utilities for task monitoring and execution, system information, symbol table management, etc.

Many of the routines here are simply command-oriented interfaces to more general routines contained elsewhere in VxWorks. Users should feel free to modify or extend this library, and may find it preferable to customize capabilities by creating a new private library, using this one as a model, and appropriately linking the new one into the system.

Some routines here have optional parameters. If those parameters are zero, which is what the shell supplies if no argument is typed, default values are typically assumed.

A number of the routines in this module take an optional task name or ID as an argument. If this argument is omitted or zero, the "current" task is used. The current task (or "default" task) is the last task referenced. The **usrLib** library uses *taskIdDefault()* to set and get the last-referenced task ID, as do many other VxWorks routines.

**NOTE** This library uses a small number of undocumented VxWorks internal routines.

**INCLUDE FILES** **usrLib.h**

**SEE ALSO** **spyLib**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## usrScsi

|          |                                                     |
|----------|-----------------------------------------------------|
| NAME     | <b>usrScsi</b> – SCSI initialization                |
| ROUTINES | <i>usrScsiConfig()</i> – configure SCSI peripherals |

---

## usrSmObj

|          |                                                          |
|----------|----------------------------------------------------------|
| NAME     | <b>usrSmObj</b> – shared memory object initialization    |
| ROUTINES | <i>usrSmObjInit()</i> – initialize shared memory objects |

---

## vmBaseLib

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NAME          | <b>vmBaseLib</b> – base virtual memory support library                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| ROUTINES      | <i>vmBaseLibInit()</i> – initialize base virtual memory support<br><i>vmBaseGlobalMapInit()</i> – initialize global mapping<br><i>vmBaseStateSet()</i> – change the state of a block of virtual memory<br><i>vmBasePageSizeGet()</i> – return the page size                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| DESCRIPTION   | <p>This library provides the minimal MMU (Memory Management Unit) support needed in a system. Its primary purpose is to create cache-safe buffers for <b>cacheLib</b>. Buffers are provided to optimize I/O throughput.</p> <p>A call to <i>vmBaseLibInit()</i> initializes this library, thus permitting <i>vmBaseGlobalMapInit()</i> to initialize the MMU and set up MMU translation tables. Additionally, <i>vmBaseStateSet()</i> can be called to change the translation tables dynamically.</p> <p>This library is a release-bundled complement to <b>vmLib</b> and <b>vmShow</b>, modules that offer full-featured MMU support and virtual memory information display routines. The <b>vmLib</b> and <b>vmShow</b> libraries are distributed as the unbundled virtual memory support option, VxVMI.</p> |
| CONFIGURATION | Bundled MMU support is included in VxWorks when the configuration macro <b>INCLUDE_MMU_BASIC</b> is defined. If the configuration macro <b>INCLUDE_MMU_FULL</b> is also defined, the default is full MMU support (unbundled).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

**vmLib**

|                      |                                                                  |
|----------------------|------------------------------------------------------------------|
| <b>INCLUDE FILES</b> | sysLib.h, vmLib.h                                                |
| <b>SEE ALSO</b>      | vmLib, vmShow, <i>VxWorks Programmer's Guide: Virtual Memory</i> |

---

## vmLib

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>vmLib</b> – architecture-independent virtual memory support library (VxVMI Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>ROUTINES</b>    | <p><i>vmLibInit()</i> – initialize the virtual memory support module (VxVMI Opt.)</p> <p><i>vmGlobalMapInit()</i> – initialize global mapping (VxVMI Opt.)</p> <p><i>vmContextCreate()</i> – create a new virtual memory context (VxVMI Opt.)</p> <p><i>vmContextDelete()</i> – delete a virtual memory context (VxVMI Opt.)</p> <p><i>vmStateSet()</i> – change the state of a block of virtual memory (VxVMI Opt.)</p> <p><i>vmStateGet()</i> – get the state of a page of virtual memory (VxVMI Opt.)</p> <p><i>vmMap()</i> – map physical space into virtual space (VxVMI Opt.)</p> <p><i>vmGlobalMap()</i> – map physical pages to virtual space in shared global virtual memory (VxVMI Opt.)</p> <p><i>vmGlobalInfoGet()</i> – get global virtual memory information (VxVMI Opt.)</p> <p><i>vmPageBlockSizeGet()</i> – get the architecture-dependent page block size (VxVMI Opt.)</p> <p><i>vmTranslate()</i> – translate a virtual address to a physical address (VxVMI Opt.)</p> <p><i>vmPageSizeGet()</i> – return the page size (VxVMI Opt.)</p> <p><i>vmCurrentGet()</i> – get the current virtual memory context (VxVMI Opt.)</p> <p><i>vmCurrentSet()</i> – set the current virtual memory context (VxVMI Opt.)</p> <p><i>vmEnable()</i> – enable or disable virtual memory (VxVMI Opt.)</p> <p><i>vmTextProtect()</i> – write-protect a text segment (VxVMI Opt.)</p> |
| <b>DESCRIPTION</b> | <p>This library provides an architecture-independent interface to the CPU's memory management unit (MMU). Although <b>vmLib</b> is implemented with architecture-specific libraries, application code need never reference directly the architecture-dependent code in these libraries.</p> <p>A fundamental goal in the design of <b>vmLib</b> was to permit transparent backward compatibility with previous versions of VxWorks that did not use the MMU. System designers may opt to disable the MMU because of timing constraints, and some architectures do not support MMUs; therefore VxWorks functionality must not be dependent on the MMU. The resulting design permits a transparent configuration with no change in the programming environment (but the addition of several protection features, such as text segment protection) and the ability to disable virtual memory in systems that require it.</p> <p>The <b>vmLib</b> library provides a mechanism for creating virtual memory contexts, <i>vmContextCreate()</i>. These contexts are not automatically created for individual tasks, but</p>                                                                                                                                                                                                                                                                |

may be created dynamically by tasks, and swapped in and out in an application specific manner.

All virtual memory contexts share a global transparent mapping of virtual to physical memory for all of local memory and the local hardware device space (defined in **sysLib.c** for each board port in the **sysPhysMemDesc** data structure). When the system is initialized, all of local physical memory is accessible at the same address in virtual memory (this is done with calls to **vmGlobalMap()**.) Modifications made to this global mapping in one virtual memory context appear in all virtual memory contexts. For example, if the exception vector table (which resides at address 0 in physical memory) is made read only by calling **vmStateSet()** on virtual address 0, the vector table will be read only in all virtual memory contexts.

Private virtual memory can also be created. When physical pages are mapped to virtual memory that is not in the global transparent region, this memory becomes accessible only in the context in which it was mapped. (The physical pages will also be accessible in the transparent translation at the physical address, unless the virtual pages in the global transparent translation region are explicitly invalidated.) State changes (writability, validity, etc.) to a section of private virtual memory in a virtual memory context do not appear in other contexts. To facilitate the allocation of regions of virtual space, **vmGlobalInfoGet()** returns a pointer to an array of booleans describing which portions of the virtual address space are devoted to global memory. Each successive array element corresponds to contiguous regions of virtual memory the size of which is architecture-dependent and which may be obtained with a call to **vmPageBlockSizeGet()**. If the boolean array element is true, the corresponding region of virtual memory, a "page block", is reserved for global virtual memory and should not be used for private virtual memory. (If **vmMap()** is called to map virtual memory previously defined as global, the routine will return an error.)

All the state information for a block of virtual memory can be set in a single call to **vmStateSet()**. It performs parameter checking and checks the validity of the specified virtual memory context. It may also be used to set architecture-dependent state information. See **vmLib.h** for additional architecture-dependent state information.

The routine **vmContextShow()** in **vmShow** displays the virtual memory context for a specified context. For more information, see the manual entry for this routine.

**CONFIGURATION** Full MMU support (**vmLib**, and optionally, **vmShow**) is included in VxWorks when the configuration macro **INCLUDE\_MMU\_FULL** is defined. If the configuration macro **INCLUDE\_MMU\_BASIC** is also defined, the default is full MMU support (unbundled).

The **sysLib.c** library contains a data structure called **sysPhysMemDesc**, which is an array of **PHYS\_MEM\_DESC** structures. Each element of the array describes a contiguous section of physical memory. The description of this memory includes its physical address, the virtual address where it should be mapped (typically, this is the same as the physical address, but not necessarily so), an initial state for the memory, and a mask defining which state bits in the state value are to be set. Default configurations are defined for each board support package (BSP), but these mappings may be changed to suit user-specific

**vmShow**

system configurations. For example, the user may need to map additional VME space where the backplane network interface data structures appear.

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b>  | This library and <b>vmShow</b> are distributed as the unbundled virtual memory support option, VxVMI. A scaled down version, <b>vmBaseLib</b> , is provided with VxWorks for systems that do not permit optional use of the MMU, or for architectures that require certain features of the MMU to perform optimally (in particular, architectures that rely heavily on caching, but do not support bus snooping, and thus require the ability to mark interprocessor communications buffers as non-cacheable.) Most routines in <b>vmBaseLib</b> are referenced internally by VxWorks; they are not callable by application code. |
| <b>INCLUDE FILES</b> | <b>vmLib.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>      | <b>sysLib</b> , <b>vmShow</b> , <i>VxWorks Programmer's Guide: Virtual Memory</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

## vmShow

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>vmShow</b> – virtual memory show routines (VxVMI Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>ROUTINES</b>      | <i>vmShowInit()</i> – include virtual memory show facility (VxVMI Opt.)<br><i>vmContextShow()</i> – display the translation table for a context (VxVMI Opt.)                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>   | <p>This library contains virtual memory information display routines.</p> <p>The routine <i>vmShowInit()</i> links this facility into the VxWorks system. It is called automatically when this facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none"> <li>– If you use the configuration header files, define both <b>INCLUDE_MMU_FULL</b> and <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</li> <li>– If you use the Tornado project facility, select <b>INCLUDE_MMU_FULL_SHOW</b>.</li> </ul> |
| <b>AVAILABILITY</b>  | This module and <b>vmLib</b> are distributed as the unbundled virtual memory support option, VxVMI.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>INCLUDE FILES</b> | <b>vmLib.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>      | <b>vmLib</b> , <i>VxWorks Programmer's Guide: Virtual Memory</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |



---

## vxLib

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>vxLib</b> – miscellaneous support routines                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>ROUTINES</b>      | <p><i>vxTas()</i> – C-callable atomic test-and-set primitive</p> <p><i>vxMemArchProbe()</i> – architecture specific part of vxMemProbe</p> <p><i>vxMemProbe()</i> – probe an address for a bus error</p> <p><i>vxMemProbeAsi()</i> – probe address in ASI space for bus error (SPARC)</p> <p><i>vxSSEnable()</i> – enable the superscalar dispatch (MC68060)</p> <p><i>vxSSDisable()</i> – disable the superscalar dispatch (MC68060)</p> <p><i>vxPowerModeSet()</i> – set the power management mode (PowerPC)</p> <p><i>vxPowerModeGet()</i> – get the power management mode (PowerPC)</p> <p><i>vxPowerDown()</i> – place the processor in reduced-power mode (PowerPC)</p> |
| <b>DESCRIPTION</b>   | This module contains miscellaneous VxWorks support routines.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>INCLUDE FILES</b> | <b>vxLib.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

---

## VXWList

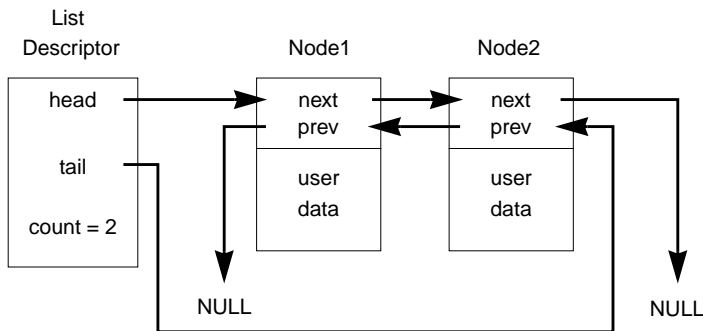
|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>    | <b>VXWList</b> – simple linked list class (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>METHODS</b> | <p><i>VXWList::VXWList()</i> – initialize a list</p> <p><i>VXWList::VXWList()</i> – initialize a list as a copy of another</p> <p><i>VXWList::~~VXWList()</i> – free up a list</p> <p><i>VXWList::add()</i> – add a node to the end of list</p> <p><i>VXWList::concat()</i> – concatenate two lists</p> <p><i>VXWList::count()</i> – report the number of nodes in a list</p> <p><i>VXWList::extract()</i> – extract a sublist from list</p> <p><i>VXWList::find()</i> – find a node in list</p> <p><i>VXWList::first()</i> – find first node in list</p> <p><i>VXWList::get()</i> – delete and return the first node from list</p> <p><i>VXWList::insert()</i> – insert a node in list after a specified node</p> <p><i>VXWList::last()</i> – find the last node in list</p> <p><i>VXWList::next()</i> – find the next node in list</p> <p><i>VXWList::nStep()</i> – find a list node <i>nStep</i> steps away from a specified node</p> <p><i>VXWList::nth()</i> – find the Nth node in a list</p> <p><i>VXWList::previous()</i> – find the previous node in list</p> <p><i>VXWList::remove()</i> – delete a specified node from list</p> |

**DESCRIPTION**

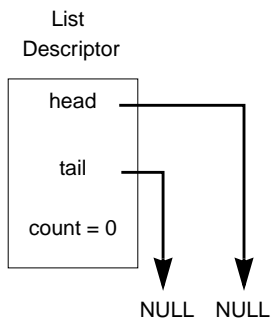
The VXWList class supports the creation and maintenance of a doubly linked list. The class contains pointers to the first and last nodes in the list, and a count of the number of nodes in the list. The nodes in the list are derived from the structure NODE, which provides two pointers: **NODE::next** and **NODE::previous**. Both the forward and backward chains are terminated with a NULL pointer.

The VXWList class simply manipulates the linked-list data structures; no kernel functions are invoked. In particular, linked lists by themselves provide no task synchronization or mutual exclusion. If multiple tasks will access a single linked list, that list must be guarded with some mutual-exclusion mechanism (such as a mutual-exclusion semaphore).

**NON-EMPTY LIST**



**EMPTY LIST**



**WARNINGS**

Use only single inheritance! This class is an interface to the VxWorks library **lstLib**. More sophisticated alternatives are available in the **Tools.h++** class libraries.

**EXAMPLE**

The following example illustrates how to create a list by deriving elements from NODE and putting them on a VXWList:

```

class myListNode : public NODE
{
public:
 myListNode ()
 {
 }
private:
};
VXWList myList;
myListNode a, b, c;
NODE * pEl = &c;
void useList ()
{
 myList.add (&a);
 myList.insert (pEl, &b);
}

```

INCLUDE FILES    vxwLstLib.h

---

## VXWMemPart

**NAME**            VXWMemPart – memory partition classes (WFC Opt.)

**METHODS**      *VXWMemPart::VXWMemPart()* – create a memory partition  
*VXWMemPart::addToPool()* – add memory to a memory partition  
*VXWMemPart::alignedAlloc()* – allocate aligned memory from partition  
*VXWMemPart::alloc()* – allocate a block of memory from partition  
*VXWMemPart::findMax()* – find the size of the largest available free block  
*VXWMemPart::free()* – free a block of memory in partition  
*VXWMemPart::info()* – get partition information  
*VXWMemPart::options()* – set the debug options for memory partition  
*VXWMemPart::realloc()* – reallocate a block of memory in partition  
*VXWMemPart::show()* – show partition blocks and statistics

**DESCRIPTION**    The **VXWMemPart** class provides core facilities for managing the allocation of blocks of memory from ranges of memory called memory partitions.

The allocation of memory, using routines such as *VXWMemPart::alloc()*, is done with a first-fit algorithm. Adjacent blocks of memory are coalesced when they are freed with *VXWMemPart::free()*. There is also a routine provided for allocating memory aligned to a specified boundary from a specific memory partition, *VXWMemPart::alignedAlloc()*.

**VXWModule**

**CAVEATS** Architectures have various alignment constraints. To provide optimal performance, `VXWMemPart::alloc()` returns a pointer to a buffer having the appropriate alignment for the architecture in use. The portion of the allocated buffer reserved for system bookkeeping, known as the overhead, may vary depending on the architecture.

| Architecture | Boundary | Overhead |
|--------------|----------|----------|
| 68K          | 4        | 8        |
| SPARC        | 8        | 12       |
| MIPS         | 8        | 12       |
| i960         | 16       | 16       |

**INCLUDE FILES** `vxwMemPartLib.h`

**SEE ALSO** `vxwSmLib`

---

## VXWModule

**NAME** `VXWModule` – object module class (WFC Opt.)

**METHODS**

- `VXWModule::VXWModule()` – build module object from module ID
- `VXWModule::VXWModule()` – load an object module at specified memory addresses
- `VXWModule::VXWModule()` – load an object module into memory
- `VXWModule::VXWModule()` – create and initialize an object module
- `VXWModule::~VXWModule()` – unload an object module
- `VXWModule::flags()` – get the flags associated with this module
- `VXWModule::info()` – get information about object module
- `VXWModule::name()` – get the name associated with module
- `VXWModule::segFirst()` – find the first segment in module
- `VXWModule::segGet()` – get (delete and return) the first segment from module
- `VXWModule::segNext()` – find the next segment in module

**DESCRIPTION** The `VXWModule` class provides a generic object-module loading facility. Any object files in a supported format may be loaded into memory, relocated properly, their external references resolved, and their external definitions added to the system symbol table for use by other modules. Modules may be loaded from any I/O stream.

**INCLUDE FILE** `vxwLoadLib.h`

**SEE ALSO** `usrLib`, `symLib`, `VXWMemPart`, *VxWorks Programmer's Guide: C++ Development*

---

# VXWMsgQ

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | VXWMsgQ – message queue classes (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>METHODS</b>     | <i>VXWMsgQ::VXWMsgQ()</i> – create and initialize a message queue<br><i>VXWMsgQ::VXWMsgQ()</i> – build message-queue object from ID<br><i>VXWMsgQ::~~VXWMsgQ()</i> – delete message queue<br><i>VXWMsgQ::send()</i> – send a message to message queue<br><i>VXWMsgQ::receive()</i> – receive a message from message queue<br><i>VXWMsgQ::numMsgs()</i> – report the number of messages queued<br><i>VXWMsgQ::info()</i> – get information about message queue<br><i>VXWMsgQ::show()</i> – show information about a message queue                            |
| <b>DESCRIPTION</b> | The <b>VXWMsgQ</b> class provides message queues, the primary intertask communication mechanism within a single CPU. Message queues allow a variable number of messages (varying in length) to be queued in first-in-first-out (FIFO) order. Any task or interrupt service routine can send messages to a message queue. Any task can receive messages from a message queue. Multiple tasks can send to and receive from the same message queue. Full-duplex communication between two tasks generally requires two message queues, one for each direction. |

## CREATING AND USING MESSAGE QUEUES

The message-queue constructor takes parameters to specify the maximum number of messages that can be queued to that message queue and the maximum length in bytes of each message. Enough buffer space is pre-allocated to accommodate the specified number of messages of specified length.

A task or interrupt service routine sends a message to a message queue with *VXWMsgQ::send()*. If no tasks are waiting for messages on the message queue, the message is simply added to the buffer of messages for that queue. If any tasks are already waiting to receive a message from the message queue, the message is immediately delivered to the first waiting task.

A task receives a message from a message queue with *VXWMsgQ::receive()*. If any messages are already available in the message queue's buffer, the first message is immediately dequeued and returned to the caller. If no messages are available, the calling task blocks and joins a queue of tasks waiting for messages. This queue of waiting tasks can be ordered either by task priority or FIFO, as specified in an option parameter when the queue is created.

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>TIMEOUTS</b> | Both <i>VXWMsgQ::send()</i> and <i>VXWMsgQ::receive()</i> take timeout parameters. When sending a message, if no buffer space is available to queue the message, the timeout specifies how many ticks to wait for space to become available. When receiving a message, the timeout specifies how many ticks to wait if no message is immediately available. The <i>timeout</i> parameter can have the special values <code>NO_WAIT</code> (0) or |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**VXWRingBuf**

WAIT\_FOREVER (-1). NO\_WAIT means the routine should return immediately; WAIT\_FOREVER means the routine should never time out.

**URGENT MESSAGES**

The `VXWMsgQ::send()` routine allows the priority of a message to be specified as either normal (`MSG_PRI_NORMAL`) or urgent (`MSG_PRI_URGENT`). Normal priority messages are added to the tail of the list of queued messages, while urgent priority messages are added to the head of the list.

**INCLUDE FILES** `vxwMsgQLib.h`

**SEE ALSO** `pipeDrv`, `msgQSmLib`, *VxWorks Programmer's Guide: Basic OS*

---

## VXWRingBuf

**NAME** `VXWRingBuf` – ring buffer class (WFC Opt.)

**METHODS**

- `VXWRingBuf::VXWRingBuf()` – create an empty ring buffer
- `VXWRingBuf::VXWRingBuf()` – build ring-buffer object from existing ID
- `VXWRingBuf::~VXWRingBuf()` – delete ring buffer
- `VXWRingBuf::get()` – get characters from ring buffer
- `VXWRingBuf::put()` – put bytes into ring buffer
- `VXWRingBuf::flush()` – make ring buffer empty
- `VXWRingBuf::freeBytes()` – determine the number of free bytes in ring buffer
- `VXWRingBuf::isEmpty()` – test whether ring buffer is empty
- `VXWRingBuf::isFull()` – test whether ring buffer is full (no more room)
- `VXWRingBuf::moveAhead()` – advance ring pointer by *n* bytes
- `VXWRingBuf::nBytes()` – determine the number of bytes in ring buffer
- `VXWRingBuf::putAhead()` – put a byte ahead in a ring buffer without moving ring pointers

**DESCRIPTION** The `VXWRingBuf` class provides routines for creating and using ring buffers, which are first-in-first-out circular buffers. The routines simply manipulate the ring buffer data structure; no kernel functions are invoked. In particular, ring buffers by themselves provide no task synchronization or mutual exclusion.

However, the ring buffer pointers are manipulated in such a way that a reader task (invoking `VXWRingBuf::get()`) and a writer task (invoking `VXWRingBuf::put()`) can access a ring simultaneously without requiring mutual exclusion. This is because readers only affect a *read* pointer and writers only affect a *write* pointer in a ring buffer data structure. However, access by multiple readers or writers *must* be interlocked through a mutual exclusion mechanism (for example, a mutual-exclusion semaphore guarding a ring buffer).

INCLUDE FILES vxwRngLib.h

1

---

## VXWSem

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | VXWSem – semaphore classes (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>METHODS</b>     | <p>VXWSem::VXWSem() – build semaphore object from semaphore ID</p> <p>VXWSem::~~VXWSem() – delete a semaphore</p> <p>VXWSem::give() – give a semaphore</p> <p>VXWSem::take() – take a semaphore</p> <p>VXWSem::flush() – unblock every task pended on a semaphore</p> <p>VXWSem::id() – reveal underlying semaphore ID</p> <p>VXWSem::info() – get a list of task IDs that are blocked on a semaphore</p> <p>VXWSem::show() – show information about a semaphore</p> <p>VXWCsem::VXWCsem() – create and initialize a counting semaphore</p> <p>VXWBsem::VXWBsem() – create and initialize a binary semaphore</p> <p>VXWMsem::VXWMsem() – create and initialize a mutual-exclusion semaphore</p> <p>VXWMsem::giveForce() – give a mutual-exclusion semaphore without restrictions</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>Semaphores are the basis for synchronization and mutual exclusion in VxWorks. They are powerful in their simplicity and form the foundation for numerous VxWorks facilities.</p> <p>Different semaphore types serve different needs, and while the behavior of the types differs, their basic interface is the same. The VXWSem class provides semaphore routines common to all VxWorks semaphore types. For all types, the two basic operations are VXWSem::take() and VXWSem::give(), the acquisition or relinquishing of a semaphore.</p> <p>Semaphore creation and initialization is handled by the following classes, which inherit the basic operations from VXWSem:</p> <ul style="list-style-type: none"> <li>VXWBsem – binary semaphores</li> <li>VXWCsem – counting semaphores</li> <li>VXWMsem – mutual exclusion semaphores</li> </ul> <p>Two additional semaphore classes provide semaphores that operate over shared memory (with the optional product VxMP). These classes also inherit from VXWSmNameLib; they are described in vxwSmLib. The following are the class names for these shared-memory semaphores:</p> <ul style="list-style-type: none"> <li>VXWSmBsem – shared-memory binary semaphores</li> <li>VXWSmCsem – shared-memory counting semaphores</li> </ul> <p>Binary semaphores offer the greatest speed and the broadest applicability.</p> <p>The VXWSem class provides all other semaphore operations, including routines for semaphore control, deletion, and information.</p> |

**SEMAPHORE CONTROL**

The `VXWSem::take()` call acquires a specified semaphore, blocking the calling task or making the semaphore unavailable. All semaphore types support a timeout on the `VXWSem::take()` operation. The timeout is specified as the number of ticks to remain blocked on the semaphore. Timeouts of `WAIT_FOREVER` and `NO_WAIT` codify common timeouts. If a `VXWSem::take()` times out, it returns `ERROR`. Refer to the library of the specific semaphore type for the exact behavior of this operation.

The `VXWSem::give()` call relinquishes a specified semaphore, unblocking a pended task or making the semaphore available. Refer to the library of the specific semaphore type for the exact behavior of this operation.

The `VXWSem::flush()` call may be used to atomically unblock all tasks pended on a semaphore queue; that is, it unblocks all tasks before any are allowed to run. It may be thought of as a broadcast operation in synchronization applications. The state of the semaphore is unchanged by the use of `VXWSem::flush()`; it is not analogous to `VXWSem::give()`.

**SEMAPHORE DELETION**

The `VXWSem::~~VXWSem()` destructor terminates a semaphore and deallocates any associated memory. The deletion of a semaphore unblocks tasks pended on that semaphore; the routines which were pended return `ERROR`. Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that already has taken (owns) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully taken.

**SEMAPHORE INFORMATION**

The `VXWSem::info()` call is a useful debugging aid, reporting all tasks blocked on a specified semaphore. It provides a snapshot of the queue at the time of the call, but because semaphores are dynamic, the information may be out of date by the time it is available. As with the current state of the semaphore, use of the queue of pended tasks should be restricted to debugging uses only.

**INCLUDE FILES**     `vxwSemLib.h`

**SEE ALSO**         `vxwTaskLib`, `vxwSmLib`, *VxWorks Programmer's Guide: Basic OS*



---

## VXWsmName

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>VXWsmName</b> – naming behavior common to all shared memory classes (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>METHODS</b>       | <p><b>VXWsmName::~~VXWsmName()</b> – remove an object from the shared memory objects name database</p> <p><b>VXWsmName::nameSet()</b> – define a name string in the shared-memory name database</p> <p><b>VXWsmName::nameGet()</b> – get name and type of a shared memory object</p> <p><b>VXWsmName::nameGet()</b> – get name of a shared memory object</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b>   | <p>This class library provides facilities for managing entries in the shared memory objects name database. The shared memory objects name database associates a name and object type with a value and makes that information available to all CPUs. A name is an arbitrary, null-terminated string. An object type is a small integer, and its value is a global (shared) ID or a global shared memory address.</p> <p>Names are added to the shared memory name database with <b>VXWsmName::VXWsmName()</b>. They are removed by <b>VXWsmName::~~VXWsmName()</b>.</p> <p>Name database contents can be viewed using <b>smNameShow()</b>.</p> <p>The maximum number of names to be entered in the database <b>SM_OBJ_MAX_NAME</b> is defined in <b>configAll.h</b>. This value is used to determine the size of a dedicated shared memory partition from which name database fields are allocated.</p> <p>The estimated memory size required for the name database can be calculated as follows:</p> $\langle \text{name database pool size} \rangle = \text{SM\_OBJ\_MAX\_NAME} * 40 \text{ (bytes)}$ <p>The display facility for the shared memory objects name database is provided by <b>smNameShow</b>.</p> |
| <b>CONFIGURATION</b> | Before routines in this library can be called, the shared memory object facility must be initialized by calling <b>usrSmObjInit()</b> , which is found in <b>src/config/usrSmObj.c</b> . This is done automatically from the root task, <b>usrRoot()</b> , in <b>usrConfig.c</b> if <b>INCLUDE_SM_OBJ</b> is defined in <b>configAll.h</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>AVAILABILITY</b>  | This module depends on code that is distributed as a component of the unbundled shared memory objects support option, VxMP.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>INCLUDE FILES</b> | <b>vwxSmNameLib.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>      | <b>smNameLib</b> , <b>smNameShow</b> , <b>vwxSmLib</b> , <b>smObjShow</b> , <b>usrSmObjInit()</b> , <i>VxWorks Programmer's Guide: Shared Memory Objects</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

## VXWSymTab

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>    | <b>VXWSymTab</b> – symbol table class (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>METHODS</b> | <p><i>VXWSymTab::VXWSymTab()</i> – create a symbol table</p> <p><i>VXWSymTab::VXWSymTab()</i> – create a symbol-table object</p> <p><i>VXWSymTab::~~VXWSymTab()</i> – delete a symbol table</p> <p><i>VXWSymTab::add()</i> – create and add symbol to a symbol table, including group number</p> <p><i>VXWSymTab::each()</i> – call a routine to examine each entry in a symbol table</p> <p><i>VXWSymTab::findByName()</i> – look up a symbol by name</p> <p><i>VXWSymTab::findByNameAndType()</i> – look up a symbol by name and type</p> <p><i>VXWSymTab::findByValue()</i> – look up a symbol by value</p> <p><i>VXWSymTab::findByValueAndType()</i> – look up a symbol by value and type</p> <p><i>VXWSymTab::remove()</i> – remove a symbol from a symbol table</p> |

**DESCRIPTION** This class library provides facilities for managing symbol tables. A symbol table associates a name and type with a value. A name is simply an arbitrary, null-terminated string. A symbol type is a small integer (typedef **SYM\_TYPE**), and its value is a character pointer. Though commonly used as the basis for object loaders, symbol tables may be used whenever efficient association of a value with a name is needed.

If you use the **VXWSymTab** class to manage symbol tables local to your own applications, the values for **SYM\_TYPE** objects are completely arbitrary; you can use whatever one-byte integers are appropriate for your application.

If the VxWorks system symbol table is configured into your target system, you can use the **VXWSymTab** class to manipulate it based on its symbol-table ID, recorded in the global **sysSymTbl**; see *VXWSymTab::VXWSymTab()* to construct an object based on this global. In the VxWorks target-resident global symbol table, the values for **SYM\_TYPE** are **N\_ABS**, **N\_TEXT**, **N\_DATA**, and **N\_BSS** (defined in **a\_out.h**); these are all even numbers, and any of them may be combined (via boolean or) with **N\_EXT** (1). These values originate in the section names for a.out object code format, but the VxWorks system symbol table uses them as symbol types across all object formats. (The VxWorks system symbol table also occasionally includes additional types, in some object formats.)

All operations on a symbol table are interlocked by means of a mutual-exclusion semaphore in the symbol table structure.

Symbols are added to a symbol table with *VXWSymTab::add()*. Each symbol in the symbol table has a name, a value, and a type. Symbols are removed from a symbol table with *VXWSymTab::remove()*.

Symbols can be accessed by either name or value. The routine *VXWSymTab::findByName()* searches the symbol table for a symbol of a specified name. The routine *VXWSymTab::findByValue()* finds the symbol with the value closest to a specified value. The routines *VXWSymTab::findByNameAndType()* and

*VXWSymTab::findByValueAndType()* allow the symbol type to be used as an additional criterion in the searches.

Symbols in the symbol table are hashed by name into a hash table for fast look-up by name, for instance with *VXWSymTab::findByName()*. The size of the hash table is specified during the creation of a symbol table. Look-ups by value, such as with *VXWSymTab::findByValue()*, must search the table linearly; these look-ups can thus be much slower.

The routine *VXWSymTab::each()* allows each symbol in the symbol table to be examined by a user-specified function.

Name clashes occur when a symbol added to a table is identical in name and type to a previously added symbol. Whether or not symbol tables can accept name clashes is set by a parameter when the symbol table is created with *VXWSymTab::VXWSymTab()*. If name clashes are not allowed, *VXWSymTab::add()* returns an error if there is an attempt to add a symbol with identical name and type. If name clashes are allowed, adding multiple symbols with the same name and type is not an error. In such cases, *VXWSymTab::findByName()* returns the value most recently added, although all versions of the symbol can be found by *VXWSymTab::each()*.

**INCLUDE FILES**      *vxwSymLib.h*

**SEE ALSO**            *vxwLoadLib*

---

## VXWTask

**NAME**                *VXWTask* – task class (WFC Opt.)

**METHODS**

- VXWTask::VXWTask()* – initialize a task object
- VXWTask::VXWTask()* – create and spawn a task
- VXWTask::VXWTask()* – initialize a task with a specified stack
- VXWTask::~~VXWTask()* – delete a task
- VXWTask::activate()* – activate a task
- VXWTask::deleteForce()* – delete a task without restriction
- VXWTask::envCreate()* – create a private environment
- VXWTask::errNo()* – retrieve error status value
- VXWTask::errNo()* – set error status value
- VXWTask::id()* – reveal task ID
- VXWTask::info()* – get information about a task
- VXWTask::isReady()* – check if task is ready to run
- VXWTask::isSuspended()* – check if task is suspended
- VXWTask::kill()* – send a signal to task
- VXWTask::name()* – get the name associated with a task ID

**VXWTask**

**VXWTask::options()** – examine task options  
**VXWTask::options()** – change task options  
**VXWTask::priority()** – examine the priority of task  
**VXWTask::priority()** – change the priority of a task  
**VXWTask::registers()** – set a task's registers  
**VXWTask::registers()** – get task registers from the TCB  
**VXWTask::restart()** – restart task  
**VXWTask::resume()** – resume task  
**VXWTask::show()** – display the contents of task registers  
**VXWTask::show()** – display task information from TCBs  
**VXWTask::sigqueue()** – send a queued signal to task  
**VXWTask::SRSet()** – set the task status register (MC680x0, MIPS, i386/i486)  
**VXWTask::statusString()** – get task status as a string  
**VXWTask::suspend()** – suspend task  
**VXWTask::tcb()** – get the task control block  
**VXWTask::varAdd()** – add a task variable to task  
**VXWTask::varDelete()** – remove a task variable from task  
**VXWTask::varGet()** – get the value of a task variable  
**VXWTask::varInfo()** – get a list of task variables  
**VXWTask::varSet()** – set the value of a task variable

**DESCRIPTION** This library provides the interface to the VxWorks task management facilities. This class library provides task control services, programmatic access to task information and debugging features, and higher-level task information display routines.

**TASK CREATION** Tasks are created with the constructor **VXWTask::VXWTask()**. Task creation consists of the following: allocation of memory for the stack and task control block (**WIND\_TCB**), initialization of the **WIND\_TCB**, and activation of the **WIND\_TCB**. Special needs may require the use of the lower-level method **VXWTask::activate()**.

Tasks in VxWorks execute in the most privileged state of the underlying architecture. In a shared address space, processor privilege offers no protection advantages and actually hinders performance.

There is no limit to the number of tasks created in VxWorks, as long as sufficient memory is available to satisfy allocation requirements.

**TASK DELETION** If a task exits its "main" routine, specified during task creation, the kernel implicitly calls **exit()** to delete the task. Tasks can be deleted with the **exit()** routine, or explicitly with the **delete** operator, which arranges to call the class destructor **VXWTask::~VXWTask()**.

Task deletion must be handled with extreme care, due to the inherent difficulties of resource reclamation. Deleting a task that owns a critical resource can cripple the system, since the resource may no longer be available. Simply returning a resource to an available state is not a viable solution, since the system can make no assumption as to the state of a particular resource at the time a task is deleted.

A task can protect itself from deletion by taking a mutual-exclusion semaphore created with the `SEM_DELETE_SAFE` option (see `vxwSemLib` for more information). Many VxWorks system resources are protected in this manner, and application designers may wish to consider this facility where dynamic task deletion is a possibility.

The `sigLib` facility may also be used to allow a task to execute clean-up code before actually expiring.

|                        |                                                                                                                                                                                                                                                                                                  |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>TASK CONTROL</b>    | The following methods control task state: <code>VXWTask::resume()</code> , <code>VXWTask::suspend()</code> , <code>VXWTask::restart()</code> , <code>VXWTask::priority()</code> , and <code>VXWTask::registers()</code> .                                                                        |
| <b>TASK SCHEDULING</b> | VxWorks schedules tasks on the basis of priority. Tasks may have priorities ranging from 0, the highest priority, to 255, the lowest priority. The priority of a task in VxWorks is dynamic, and an existing task's priority can be changed or examined using <code>VXWTask::priority()</code> . |
| <b>INCLUDE FILES</b>   | <code>taskLib.h</code>                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>        | <code>taskLib</code> , <code>taskHookLib</code> , <code>vxwSemLib</code> , <code>kernelLib</code> , <i>VxWorks Programmer's Guide: Basic OS</i>                                                                                                                                                  |

---

## VXWwD

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | VXWwD – watchdog timer class (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>METHODS</b>     | <p><code>VXWwD::VXWwD()</code> – construct a watchdog timer</p> <p><code>VXWwD::~VXWwD()</code> – construct a watchdog timer</p> <p><code>VXWwD::~~VXWwD()</code> – destroy a watchdog timer</p> <p><code>VXWwD::cancel()</code> – cancel a currently counting watchdog</p> <p><code>VXWwD::start()</code> – start a watchdog timer</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>This library provides a general watchdog timer facility. Any task may create a watchdog timer and use it to run a specified routine in the context of the system-clock ISR, after a specified delay.</p> <p>Once a timer has been created, it can be started with <code>VXWwD::start()</code>. The <code>VXWwD::start()</code> routine specifies what routine to run, a parameter for that routine, and the amount of time (in ticks) before the routine is to be called. (The timeout value is in ticks as determined by the system clock; see <code>sysClkRateSet()</code> for more information.) After the specified delay ticks have elapsed (unless <code>VXWwD::cancel()</code> is called first to cancel the timer) the timeout routine is invoked with the parameter specified in the <code>VXWwD::start()</code> call. The timeout routine is invoked whether the task which started the watchdog is running, suspended, or deleted.</p> |

The timeout routine executes only once per `VXWwd::start()` invocation; there is no need to cancel a timer with `VXWwd::cancel()` after it has expired, or in the expiration callback itself.

Note that the timeout routine is invoked at interrupt level, rather than in the context of the task. Thus, there are restrictions on what the routine may do. Watchdog routines are constrained to the same rules as interrupt service routines. For example, they may not take semaphores, issue other calls that may block, or use I/O system routines like `printf()`.

**EXAMPLE**

In the fragment below, if `maybeSlowRoutine()` takes more than 60 ticks, `logMsg()` will be called with the string as a parameter, causing the message to be printed on the console. Normally, of course, more significant corrective action would be taken.

```
VXWwd *pWd = new VXWwd;
pWd->start (60, logMsg, "Help, I've timed out!");
maybeSlowRoutine (); /* user-supplied routine */
delete pWd;
```

**INCLUDE FILES** `vxwWdLib.h`

**SEE ALSO** `wdLib`, `logLib`, *VxWorks Programmer's Guide: Basic OS, C++ Development*

---

## wd33c93Lib

**NAME**

`wd33c93Lib` – WD33C93 SCSI-Bus Interface Controller (SBIC) library

**ROUTINES**

`wd33c93CtrlInit()` – initialize the user-specified fields in an SBIC structure  
`wd33c93Show()` – display the values of all readable WD33C93 chip registers

**DESCRIPTION**

This library contains the main interface routines to the Western Digital WD33C93 and WD33C93A SCSI-Bus Interface Controllers (SBIC). However, these routines simply switch the calls to either the SCSI-1 or SCSI-2 drivers, implemented in `wd33c93Lib1` and `wd33c93Lib2` respectively, as configured by the Board Support Package (BSP).

In order to configure the SCSI-1 driver, which depends upon `scsi1Lib`, the `wd33c93CtrlCreate()` routine, defined in `wd33c93Lib1`, must be invoked. Similarly, `wd33c93CtrlCreateScsi2()`, defined in `wd33c93Lib2` and dependent on `scsi2Lib`, must be called to configure and initialize the SCSI-2 driver.

**INCLUDE FILES**

`wd33c93.h`, `wd33c93_1.h`, `wd33c93_2.h`

**SEE ALSO**

`scsiLib`, `scsi1Lib`, `scsi2Lib`, `wd33c93Lib1`, `wd33c93Lib2`, *Western Digital WD33C92/93 SCSI-Bus Interface Controller, Western Digital WD33C92A/93A SCSI-Bus Interface Controller, VxWorks Programmer's Guide: I/O System*

---

## wd33c93Lib1

- NAME** **wd33c93Lib1** – WD33C93 SCSI-Bus Interface Controller library (SCSI-1)
- ROUTINES** *wd33c93CtrlCreate()* – create and partially initialize a WD33C93 SBIC structure
- DESCRIPTION** This library contains part of the I/O driver for the Western Digital WD33C93 and WD33C93A SCSI-Bus Interface Controllers (SBIC). The driver routines in this library depend on the SCSI-1 version of the SCSI standard; for driver routines that do not depend on SCSI-1 or SCSI-2, and for overall SBIC driver documentation, see **wd33c93Lib**.
- USER-CALLABLE ROUTINES**
- Most of the routines in this driver are accessible only through the I/O system. The only exception in this portion of the driver is *wd33c93CtrlCreate()*, which creates a controller structure.
- INCLUDE FILES** **wd33c93.h**, **wd33c93\_1.h**
- SEE ALSO** **scsiLib**, **scsi1Lib**, **wd33c93Lib**

---

## wd33c93Lib2

- NAME** **wd33c93Lib2** – WD33C93 SCSI-Bus Interface Controller library (SCSI-2)
- ROUTINES** *wd33c93CtrlCreateScsi2()* – create and partially initialize an SBIC structure
- DESCRIPTION** This library contains part of the I/O driver for the Western Digital WD33C93 family of SCSI-2 Bus Interface Controllers (SBIC). It is designed to work with **scsi2Lib**. The driver routines in this library depend on the SCSI-2 ANSI specification; for general driver routines and for overall SBIC documentation, see **wd33c93Lib**.
- USER-CALLABLE ROUTINES**
- Most of the routines in this driver are accessible only through the I/O system. The only exception in this portion of the driver is *wd33c93CtrlCreateScsi2()*, which creates a controller structure.
- INCLUDE FILES** **wd33c93.h**, **wd33c93\_2.h**
- SEE ALSO** **scsiLib**, **scsi2Lib**, **wd33c93Lib**, *VxWorks Programmer's Guide: I/O System*

## wdbEndPktDrv

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>           | <b>wdbEndPktDrv</b> – END based packet driver for lightweight UDP/IP                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>ROUTINES</b>       | No Callable Routines                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>    | This is an END based driver for the WDB system. It uses the MUX and END based drivers to allow for interaction between the target and target server.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>USAGE</b>          | The driver is typically only called only from <b>usrWdb.c</b> . The only directly callable routine in this module is <i>wdbEndPktDevInit()</i> . Your <b>configAll.h</b> file will have to be modified so that <b>WDB_COMM_TYPE</b> is defined as <b>WDB_COMM_END</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>DATA BUFFERING</b> | <p>The drivers only need to handle one input packet at a time because the WDB protocol only supports one outstanding host-request at a time. If multiple input packets arrive, the driver can simply drop them. The driver then loans the input buffer to the WDB agent, and the agent invokes a driver callback when it is done with the buffer.</p> <p>For output, the agent will pass the driver a chain of mbufs, which the driver must send as a packet. When it is done with the mbufs, it calls <i>wdbMbufChainFree()</i> to free them. The header file <b>wdbMbufLib.h</b> provides the calls for allocating, freeing, and initializing mbufs for use with the lightweight UDP/IP interpreter. It ultimately makes calls to the routines <i>wdbMbufAlloc</i> and <i>wdbMbufFree</i>, which are provided in source code in <b>usrWdb.c</b>.</p> |

---

## wdbLib

|                      |                                                                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>wdbLib</b> – WDB agent context management library                                                                                                                                                                                                                                                                |
| <b>ROUTINES</b>      | <i>wdbSystemSuspend()</i> – suspend the system.                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b>   | This library provides a routine to transfer control from the run time system to the WDB agent running in external mode. This agent in external mode allows a system-wide control, including ISR debugging, from a host tool (e.g.: Crosswind, WindSh ...) through the target server and the WDB communication link. |
| <b>INCLUDE FILES</b> | <b>wdb/wdbLib.h</b>                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>      | <i>API Guide: WTX Protocol</i> , <i>Tornado User's Guide: Overview</i>                                                                                                                                                                                                                                              |



---

## wdbNetromPktDrv

|                    |                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>wdbNetromPktDrv</b> – NETROM packet driver for the WDB agent                                                                                                                                                                                                                                                              |
| <b>ROUTINES</b>    | <i>wdbNetromPktDevInit()</i> – initialize a NETROM packet device for the WDB agent                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | This is a lightweight NETROM driver that interfaces with the WDB agent's UDP/IP interpreter. It allows the WDB agent to communicate with the host using the NETROM ROM emulator. It uses the emulator's read-only protocol for bi-directional communication. It requires that NetROM's <code>udpsrcmode</code> option is on. |

---

## wdbPipePktDrv

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>wdbPipePktDrv</b> – pipe packet driver for lightweight UDP/IP                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>ROUTINES</b>    | <i>wdbPipePktDevInit()</i> – initialize a pipe packet device.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>DESCRIPTION</b> | <p>This module is a pipe for drivers interfacing with the WDB agent's lightweight UDP/IP interpreter. It can be used as a starting point when writing new drivers. Such drivers are the lightweight equivalent of a network interface driver.</p> <p>These drivers, along with the lightweight UDP-IP interpreter, have two benefits over the stand combination of a netif driver + the full VxWorks networking stack; First, they can run in a much smaller amount of target memory because the lightweight UDP-IP interpreter is much smaller than the VxWorks network stack (about 800 bytes total). Second, they provide a communication path which is independant of the OS, and thus can be used to support an external mode (e.g., monitor style) debug agent.</p> <p>Throughout this file the word "pipe" is used in place of a real driver name. For example, if you were writing a lightweight driver for the lance ethernet chip, you would want to substitute "pipe" with "ln" throughout this file.</p> |

### PACKET READY CALLBACK

When the driver detects that a packet has arrived (either in its receiver ISR or in its poll input routine), it invokes a callback to pass the data to the debug agent. Right now the callback routine is called "udpRcv", however other callbacks may be added in the future. The driver's *wdbPipeDevInit()* routine should be passed the callback as a parameter and place it in the device data structure. That way the driver will continue to work if new callbacks are added later.

## MODES

Ideally the driver should support both polled and interrupt mode, and be capable of switching modes dynamically. However this is not required. When the agent is not running, the driver will be placed in "interrupt mode" so that the agent can be activated as soon as a packet arrives. If your driver does not support an interrupt mode, you can simulate this mode by spawning a VxWorks task to poll the device at periodic intervals and simulate a receiver ISR when a packet arrives.

For dynamically mode switchable drivers, be aware that the driver may be asked to switch modes in the middle of its input ISR. A driver's input ISR will look something like this:

```
doSomeStuff();
pPktDev->wdbDrvIf.stackRcv (pMbuf); /* invoke the callback */
doMoreStuff();
```

If this channel is used as a communication path to an external mode debug agent, then the agent's callback will lock interrupts, switch the device to polled mode, and use the device in polled mode for awhile. Later on the agent will unlock interrupts, switch the device back to interrupt mode, and return to the ISR. In particular, the callback can cause two mode switches, first to polled mode and then back to interrupt mode, before it returns. This may require careful ordering of the callback within the interrupt handler. For example, you may need to acknowledge the interrupt within the *doSomeStuff()* processing rather than the *doMoreStuff()* processing.

## USAGE

The driver is typically only called only from **usrWdb.c**. The only directly callable routine in this module is *wdbPipePktDevInit()*. You will need to modify **usrWdb.c** to allow your driver to be initialized by the debug agent. You will want to modify **usrWdb.c** to include your driver's header file, which should contain a definition of **WDB\_PIPE\_PKT\_MTU**. There is a default user-selectable macro called **WDB\_MTU**, which must be no larger than **WDB\_PIPE\_PKT\_MTU**. Modify the beginning of **usrWdb.c** to insure that this is the case by copying the way it is done for the other drivers. The routine *wdbCommIfInit()* also needs to be modified so that if your driver is selected as the **WDB\_COMM\_TYPE**, then your drivers init routine will be called. Search **usrWdb.c** for the macro **WDB\_COMM\_CUSTOM** and mimic that style of initialization for your driver.

## DATA BUFFERING

The drivers only need to handle one input packet at a time because the WDB protocol only supports one outstanding host-request at a time. If multiple input packets arrive, the driver can simply drop them. The driver then loans the input buffer to the WDB agent, and the agent invokes a driver callback when it is done with the buffer.

For output, the agent will pass the driver a chain of mbufs, which the driver must send as a packet. When it is done with the mbufs, it calls *wdbMbufChainFree()* to free them. The header file **wdbMbufLib.h** provides the calls for allocating, freeing, and initializing mbufs for use with the lightweight UDP/IP interpreter. It ultimately makes calls to the routines *wdbMbufAlloc()* and *wdbMbufFree()*, which are provided in source code in **usrWdb.c**.

---

## wdbSlipPktDrv

- NAME** **wdbSlipPktDrv** – a serial line packetizer for the WDB agent
- ROUTINES** *wdbSlipPktDevInit()* – initialize a SLIP packet device for a WDB agent
- DESCRIPTION** This is a lightweight SLIP driver that interfaces with the WDB agents UDP/IP interpreter. It is the lightweight equivalent of the VxWorks SLIP netif driver, and uses the same protocol to assemble serial characters into IP datagrams (namely the SLIP protocol). SLIP is a simple protocol that uses four token characters to delimit each packet:
- **FRAME\_END** (0300)
  - **FRAME\_ESC** (0333)
  - **FRAME\_TRANS\_END** (0334)
  - **FRAME\_TRANS\_ESC** (0335)
- The END character denotes the end of an IP packet. The ESC character is used with **TRANS\_END** and **TRANS\_ESC** to circumvent potential occurrences of END or ESC within a packet. If the END character is to be embedded, SLIP sends "ESC **TRANS\_END**" to avoid confusion between a SLIP-specific END and actual data whose value is END. If the ESC character is to be embedded, then SLIP sends "ESC **TRANS\_ESC**" to avoid confusion. (Note that the SLIP ESC is not the same as the ASCII ESC.)
- On the receiving side of the connection, SLIP uses the opposite actions to decode the SLIP packets. Whenever an END character is received, SLIP assumes a full packet has been received and sends on.
- This driver has an MTU of 1006 bytes. If the host is using a real SLIP driver with a smaller MTU, then you will need to lower the definition of **WDB\_MTU** in **configAll.h** so that the host and target MTU match. If you are not using a SLIP driver on the host, but instead are using the target server's wdbserial backend to connect to the agent, then you do not need to worry about incompatibilities between the host and target MTUs.

---

## wdbTsfsDrv

- NAME** **wdbTsfsDrv** – virtual generic file I/O driver for the WDB agent
- ROUTINES** *wdbTsfsDrv()* – initialize the TSFS device driver for a WDB agent
- DESCRIPTION** This library provides a virtual file I/O driver for use with the WDB agent. I/O is performed on this virtual I/O device exactly as it would be on any device referencing a VxWorks file system. File operations, such as *read()* and *write()*, move data over a

virtual I/O channel created between the WDB agent and the Tornado target server. The operations are then executed on the host file system. Because file operations are actually performed on the host file system by the target server, the file system presented by this virtual I/O device is known as the target-server file system, or TSFS.

The driver is installed with `wdbTsfsDrv()`, creating a device typically called `/tgtsvr`. See the manual page for `wdbTsfsDrv()` for more information about using this function. The initialization is done automatically, enabling access to TSFS, when `INCLUDE_WDB_TSFS` is defined. The target server also must have TSFS enabled in order to use TSFS. See the *WindView User's Guide: Data Upload* and the target server documentation.

**TSFS SOCKETS**

TSFS provides all of the functionality of other VxWorks file systems. For details, see the *VxWorks Programmer's Guide: I/O System and Local File Systems*. In addition to normal files, however, TSFS also provides basic access to TCP sockets. This includes opening the client side of a TCP socket, reading, writing, and closing the socket. Basic `setsockopt()` commands are also supported.

To open a TCP socket using TSFS, use a filename of the form:

```
TCP:server_name | server_ip:port_number
```

To open and connect a TCP socket to a server socket located on a server named `mongoose`, listening on port 2010, use the following:

```
fd = open ("/tgtsvr/TCP:mongoose:2010", 0, 0)
```

The open flags and permission arguments to the open call are ignored when opening a socket through TSFS. If the server `mongoose` has an IP number of `144.12.44.12`, you can use the following equivalent form of the command:

```
fd = open ("/tgtsvr/TCP:144.12.44.12:2010", 0, 0)
```

**DIRECTORIES**

All directory functions, such as `mkdir()`, `rmdir()`, `opendir()`, `readdir()`, `closedir()`, and `rewinddir()` are supported by TSFS, regardless of whether the target server providing TSFS is being run on a UNIX or Windows host.

While it is possible to open and close directories using `open()` and `close()`, it is not possible to read from a directory using `read()`. Instead, `readdir()` must be used. It is also not possible to write to an open directory, and opening a directory for anything other than read-only results in an error, with `errno` set to `EISDIR`. Calling `read()` on a directory returns `ERROR` with `errno` set to `EISDIR`.

**OPEN FLAGS**

When the target server that is providing the TSFS is running on a Windows host, the default file-translation mode is binary translation. If text translation is required, then `WDB_TSFS_O_TEXT` can be included in the mode argument to `open()`. For example:

```
fd = open ("/tgtsvr/foo", O_CREAT | O_RDWR | WDB_TSFS_O_TEXT, 0777)
```

If the target server providing TSFS services is running on a UNIX host, `WDB_TSFS_O_TEXT` is ignored.

**TGTSVR** For general information on the target server, see the reference entry for **tgtsvr**. In order to use this library, the target server must support and be configured with the following options:

**-R** *root*

Specify the root of the host's file system that is visible to target processes using TSFS. This flag is required to use TSFS. Files under this root are by default read only. To allow read/write access, specify **-RW**.

**-RW**

Allow read and write access to host files by target processes using TSFS. When this option is specified, access to the target server is restricted as if **-L** were also specified.

**IOCTL SUPPORT** TSFS supports the following *ioctl()* functions for controlling files and sockets. Details about each function can be found in the documentation listed below.

**FIOSEEK**

**FIOWHERE**

**FIOMKDIR**

Create a directory. The path, in this case **/tgtsvr/tmp**, must be an absolute path prefixed with the device name. To create the directory **/tmp** on the root of the TSFS file system use the following:

```
status = ioctl (fd, FIOMKDIR, "/tgtsvr/tmp")
```

**FIORMDIR**

Remove a directory. The path, in this case **/tgtsvr/foo**, must be an absolute path prefixed with the device name. To remove the directory **/foo** from the root of the TSFS file system, use the following:

```
status = ioctl (fd, FIORMDIR, "/tgtsvr/foo")
```

**FIORENAME**

Rename the file or directory represented by **fd** to the name in the string pointed to by **arg**. The path indicated by **arg** may be prefixed with the device name or not. Using this *ioctl()* function with the path **/foo/goop** produces the same outcome as the path **/tgtsvr/foo/goop**. The path is not modified to account for the current working directory, and therefore must be an absolute path.

```
char *arg = "/tgtsvr/foo/goop";
status = ioctl (fd, FIORENAME, arg);
```

**FIOREADDIR**

**FIONREAD**

Return the number of bytes ready to read on a TSFS socket file descriptor.

**FIOFSTATGET**

**FIOGETFL**

The following *ioctl()* functions can be used only on socket file descriptors. Using these functions with *ioctl()* provides similar behavior to the *setsockopt()* and *getsockopt()*

functions usually used with socket descriptors. Each command's name is derived from a *getsockopt()/setsockopt()* command and works in exactly the same way as the respective *getsockopt()/setsockopt()* command. The functions *setsockopt()* and *getsockopt()* can not be used with TSFS socket file descriptors.

For example, to enable recording of debugging information on the TSFS socket file descriptor, call:

```
int arg = 1;
status = ioctl (fd, SO_SETDEBUG, arg);
```

To determine whether recording of debugging information for the TSFS-socket file descriptor is enabled or disabled, call:

```
int arg;
status = ioctl (fd, SO_GETDEBUG, & arg);
```

After the call to *ioctl()*, *arg* contains the state of the debugging attribute.

The *ioctl()* functions supported for TSFS sockets are:

**SO\_SETDEBUG**

Equivalent to *setsockopt()* with the *SO\_DEBUG* command.

**SO\_GETDEBUG**

Equivalent to *getsockopt()* with the *SO\_DEBUG* command.

**SO\_SETSNDBUF**

This command changes the size of the send buffer of the host socket. The configuration of the WDB channel between the host and target also affects the number of bytes that can be written to the TSFS file descriptor in a single attempt.

**SO\_SETRCVBUF**

This command changes the size of the receive buffer of the host socket. The configuration of the WDB channel between the host and target also affects the number of bytes that can be read from the TSFS file descriptor in a single attempt.

**SO\_SETDONTROUTE**

Equivalent to *setsockopt()* with the *SO\_DONTROUTE* command.

**SO\_GETDONTROUTE**

Equivalent to *getsockopt()* with the *SO\_DONTROUTE* command.

**SO\_SETOOBINLINE**

Equivalent to *setsockopt()* with the *SO\_OOBINLINE* command.

**SO\_GETOOBINLINE**

Equivalent to *getsockopt()* with the *SO\_OOBINLINE* command.

**SO\_SNDURGB**

The *SO\_SNDURGB* command sends one out-of-band byte (pointed to by *arg*) through the socket.

**ERROR CODES** The routines in this library return the VxWorks error codes that most closely match the `errno`s generated by the corresponding host function. If an error is encountered that is due to a WDB failure, a WDB error is returned instead of the standard VxWorks **errno**. If an **errno** generated on the host has no reasonable VxWorks counterpart, the host **errno** is passed to the target calling routine unchanged.

**SEE ALSO** *Tornado User's Guide, VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## wdbUlipPktDrv

**NAME** **wdbUlipPktDrv** – WDB communication interface for the ULIP driver

**ROUTINES** *wdbUlipPktDevInit()* – initialize the WDB agent's communication functions for ULIP

**DESCRIPTION** This is a lightweight ULIP driver that interfaces with the WDB agent's UDP/IP interpreter. It is the lightweight equivalent of the ULIP netif driver. It provides a communication path which supports both a task mode and an external mode WDB agent.

---

## wdbUserEvtLib

**NAME** **wdbUserEvtLib** – WDB user event library

**ROUTINES** *wdbUserEvtLibInit()* – include the WDB user event library  
*wdbUserEvtPost()* – post a user event string to host tools.

**DESCRIPTION** This library contains routines for sending WDB User Events. The event is sent through the WDB agent, the WDB communication link and the target server to the host tools that have registered for it. The event received by host tools will be a WTX user event string.

**INCLUDE FILES** **wdb/wdbLib.h**

**SEE ALSO** *Tornado API Programmer's Guide: WTX Protocol*

## wdbVioDrv

**NAME** **wdbVioDrv** – virtual tty I/O driver for the WDB agent

**ROUTINES** **wdbVioDrv()** – initialize the tty driver for a WDB agent

**DESCRIPTION** This library provides a psuedo-tty driver for use with the WDB debug agent. I/O is performed on a virtual I/O device just like it is on a VxWorks serial device. The difference is that the data is not moved over a physical serial channel, but rather over a virtual channel created between the WDB debug agent and the Tornado host tools.

The driver is installed with **wdbVioDrv()**. Virtual I/O channels are created by opening the device (see **wdbVioDrv()** for details). The virtual I/O channels are defined as follows:

| Channel      | Usage                           |
|--------------|---------------------------------|
| 0            | Virtual console                 |
| 1-0xfffff    | Dynamically created on the host |
| >= 0x1000000 | User defined                    |

Once data is written to a virtual I/O channel on the target, it is sent to the host-based target server. The target server allows this data to be sent to another host tool, redirected to the "virtual console," or redirected to a file. For details see the *Tornado User's Guide*.

---

## wdLib

**NAME** **wdLib** – watchdog timer library

**ROUTINES** **wdCreate()** – create a watchdog timer  
**wdDelete()** – delete a watchdog timer  
**wdStart()** – start a watchdog timer  
**wdCancel()** – cancel a currently counting watchdog

**DESCRIPTION** This library provides a general watchdog timer facility. Any task may create a watchdog timer and use it to run a specified routine in the context of the system-clock ISR, after a specified delay.

Once a timer has been created with **wdCreate()**, it can be started with **wdStart()**. The **wdStart()** routine specifies what routine to run, a parameter for that routine, and the amount of time (in ticks) before the routine is to be called. (The timeout value is in ticks as determined by the system clock; see **sysClkRateSet()** for more information.) After the specified delay ticks have elapsed (unless **wdCancel()** is called first to cancel the timer) the timeout routine is invoked with the parameter specified in the **wdStart()** call. The



timeout routine is invoked whether the task which started the watchdog is running, suspended, or deleted.

The timeout routine executes only once per *wdStart()* invocation; there is no need to cancel a timer with *wdCancel()* after it has expired, or in the expiration callback itself.

Note that the timeout routine is invoked at interrupt level, rather than in the context of the task. Thus, there are restrictions on what the routine may do. Watchdog routines are constrained to the same rules as interrupt service routines. For example, they may not take semaphores, issue other calls that may block, or use I/O system routines like *printf()*.

**EXAMPLE**

In the fragment below, if *maybeSlowRoutine()* takes more than 60 ticks, *logMsg()* will be called with the string as a parameter, causing the message to be printed on the console. Normally, of course, more significant corrective action would be taken.

```
WDOG_ID wid = wdCreate ();
wdStart (wid, 60, logMsg, "Help, I've timed out!");
maybeSlowRoutine (); /* user-supplied routine */
wdCancel (wid);
```

**INCLUDE FILES**    **wdLib.h**

**SEE ALSO**        **logLib**, *VxWorks Programmer's Guide: Basic OS*

---

## wdShow

**NAME**            **wdShow** – watchdog show routines

**ROUTINES**        *wdShowInit()* – initialize the watchdog show facility  
*wdShow()* – show information about a watchdog

**DESCRIPTION**    This library provides routines to show watchdog statistics, such as watchdog activity, a watchdog routine, etc.

The routine *wdShowInit()* links the watchdog show facility into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:

- If you use configuration header files, define **INCLUDE\_SHOW\_ROUTINES** in **config.h**.
- If you use the Tornado project facility, select **INCLUDE\_WATCHDOGS\_SHOW**.

**INCLUDE FILES**    **wdLib.h**

**SEE ALSO**        **wdLib**, *VxWorks Programmer's Guide: Basic OS, Target Shell, windsh, Tornado User's Guide: Shell*

---

## winSio

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>winSio</b> – win serial driver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>ROUTINES</b>      | <i>winDevInit()</i> – initialize a <b>WIN_CHAN</b><br><i>winDevInit2()</i> – initialize a <b>WIN_CHAN</b> , part 2<br><i>winIntRcv()</i> – handle a channel’s receive-character interrupt<br><i>winIntTx()</i> – transmit a single character.<br><i>dummyCallback()</i> – dummy callback routine                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>   | This is the console serial driver for the Windows simulator. It receives character interrupts from Windows and sends them to VxWorks. Device data structures are defined in the header file <b>h/drv/sio/winSio.h</b> . A device data structure, <b>WIN_CHAN</b> , is defined for each channel.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>USAGE</b>         | <p>The driver is typically only called only by the BSP. The directly callable routines in this module are <i>winDevInit()</i>, <i>winDevInit2()</i>, <i>winIntRcv()</i>, and <i>winIntTx()</i>.</p> <p>The BSP calls <i>winDevInit()</i> to initialize or reset the device. It connects the driver’s interrupt handlers (<i>winIntRcv</i> and <i>winIntTx</i>) using <i>intConnect()</i>. After connecting the interrupt handlers, the BSP calls <i>winDevInit2()</i> to inform the driver that interrupt mode operation is now possible.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>BSP</b>           | <p>By convention all the BSP-specific serial initialization is performed in a file called <b>sysSerial.c</b>, which is #include’ed by <b>sysLib.c</b>. <b>sysSerial.c</b> implements at least four functions, <i>sysSerialHwInit()</i>, <i>sysSerialHwInit2()</i>, <i>sysSerialChanGet()</i>, and <i>sysSerialReset()</i>, which work as follows:</p> <p><i>sysSerialHwInit()</i> is called by <i>sysHwInit()</i> to initialize the serial devices. This routine will initialize all the board specific fields in the <b>WIN_CHAN</b> structure (e.g., register I/O addresses, etc.) before calling <i>winDevInit()</i>, which resets the device and installs the driver function pointers. <i>sysSerialHwInit()</i> should also perform any other processing needed for the serial drivers, such as configuring on-board interrupt controllers as appropriate.</p> <p><i>sysSerialHwInit2()</i> is called by <i>sysHwInit2()</i> to connect the serial driver’s interrupt handlers using <i>intConnect()</i>. After connecting the interrupt handlers, the call to <i>winDevInit2()</i> is made to permit interrupt mode operations to begin.</p> <p><i>sysSerialChanGet()</i> is called by <i>usrRoot()</i> to get the serial channel descriptor associated with a serial channel number. The routine takes a single parameter which is a channel number ranging between zero and <b>NUM_TTY</b>. It returns a pointer to the corresponding channel descriptor, <b>SIO_CHAN *</b>, which is just the address of the <b>WIN_CHAN</b> structure.</p> <p><i>sysSerialReset()</i> is called from <i>sysToMonitor()</i> and should reset the serial devices to an inactive state.</p> |
| <b>INCLUDE FILES</b> | <b>drv/sio/winSio.h</b> , <b>sioLib.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

---

## z8530Sio

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>z8530Sio</b> – Z8530 SCC Serial Communications Controller driver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>ROUTINES</b>    | <i>z8530DevInit()</i> – initialize a <b>Z8530_DUSART</b><br><i>z8530IntWr()</i> – handle a transmitter interrupt<br><i>z8530IntRd()</i> – handle a receiver interrupt<br><i>z8530IntEx()</i> – handle error interrupts<br><i>z8530Int()</i> – handle all interrupts in one vector                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | This is the driver for the Z8530 SCC (Serial Communications Controller). It uses the SCCs in asynchronous mode only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>USAGE</b>       | <p>A <b>Z8530_DUSART</b> structure is used to describe the chip. This data structure contains two <b>Z8530_CHAN</b> structures which describe the chip's two serial channels. Supported baud rates range from 50 to 38400. The default baud rate is <b>Z8530_DEFAULT_BAUD</b> (9600). The BSP may redefine this.</p> <p>The BSP's <i>sysHwInit()</i> routine typically calls <i>sysSerialHwInit()</i> which initializes all the values in the <b>Z8530_DUSART</b> structure (except the <b>SIO_DRV_FUNCS</b>) before calling <i>z8530DevInit()</i>.</p> <p>The BSP's <i>sysHwInit2()</i> routine typically calls <i>sysSerialHwInit2()</i> which connects the chips interrupts via <i>intConnect()</i> (either the single interrupt <b>z8530Int</b> or the three interrupts <b>z8530IntWr</b>, <b>z8530IntRd</b>, and <b>z8530IntEx</b>).</p> <p>This driver handles setting of hardware options such as parity (odd, even) and number of data bits (5, 6, 7, 8). Hardware flow control is provided with the signals CTS on transmit and DSR on read. Refer to the target documentation for the RS232 port configuration. The function HUPCL (hang up on last close) is supported. Default hardware options are defined by <b>Z8530_DEFAULT_OPTIONS</b>. The BSP may redefine them.</p> <p>All device registers are accessed via BSP-defined macros so that memory-mapped as well as I/O space accesses can be supported. The BSP may redefine the <b>REG_8530_READ</b> and <b>REG_8530_WRITE</b> macros as needed. By default, they are defined as simple memory-mapped accesses.</p> <p>The BSP may define <b>DATA_REG_8530_DIRECT</b> to cause direct access to the Z8530 data register, where hardware permits it. By default, it is not defined.</p> <p>The BSP may redefine the macro for the channel reset delay <b>Z8530_RESET_DELAY</b> as well as the channel reset delay counter value <b>Z8530_RESET_DELAY_COUNT</b> as required. The delay is defined as the minimum time between successive chip accesses (6 PCLKs + 200 nSec for a Z8530, 4 PCLKs for a Z85C30 or Z85230) plus an additional 4 PCLKs. At a typical PCLK frequency of 10 MHz, each PCLK is 100 nSec, giving a minimum reset delay of:</p> |

**zbufLib**

**Z8530** 10 PCLKs + 200 nSec = 1200 nSec = 1.2 uSec  
 Z85x30: 8 PCLKs = 800 nSec = 0.8 uSec

**INCLUDE FILES** `drv/sio/z8530Sio.h`

---

## **zbufLib**

**NAME** `zbufLib` – zbuf interface library

**ROUTINES**

- `zbufCreate()` – create an empty zbuf
- `zbufDelete()` – delete a zbuf
- `zbufInsert()` – insert a zbuf into another zbuf
- `zbufInsertBuf()` – create a zbuf segment from a buffer and insert into a zbuf
- `zbufInsertCopy()` – copy buffer data into a zbuf
- `zbufExtractCopy()` – copy data from a zbuf to a buffer
- `zbufCut()` – delete bytes from a zbuf
- `zbufSplit()` – split a zbuf into two separate zbufs
- `zbufDup()` – duplicate a zbuf
- `zbufLength()` – determine the length in bytes of a zbuf
- `zbufSegFind()` – find the zbuf segment containing a specified byte location
- `zbufSegNext()` – get the next segment in a zbuf
- `zbufSegPrev()` – get the previous segment in a zbuf
- `zbufSegData()` – determine the location of data in a zbuf segment
- `zbufSegLength()` – determine the length of a zbuf segment

**DESCRIPTION** This library contains routines to create, build, manipulate, and delete zbufs. Zbufs, also known as "zero copy buffers," are a data abstraction designed to allow software modules to share buffers without unnecessarily copying data.

To support the data abstraction, the subroutines in this library hide the implementation details of zbufs. This also maintains the library's independence from any particular implementation mechanism, permitting the zbuf interface to be used with other buffering schemes eventually.

Zbufs have three essential properties. First, a zbuf holds a sequence of bytes. Second, these bytes are organized into one or more segments of contiguous data, although the successive segments themselves are not usually contiguous. Third, the data within a segment may be shared with other segments; that is, the data may be in use by more than one zbuf at a time.

**ZBUF TYPES** The following data types are used in managing zbufs:

**ZBUF\_ID**

An arbitrary (but unique) integer that identifies a particular zbuf.

**ZBUF\_SEG**

An arbitrary (but unique within a single zbuf) integer that identifies a segment within a zbuf.

**ADDRESSING BYTES IN ZBUFS**

The bytes in a zbuf are addressed by the combination *zbufSeg*, *offset*. The *offset* may be positive or negative, and is simply the number of bytes from the beginning of the segment *zbufSeg*.

A *zbufSeg* can be specified as `NULL`, to identify the segment at the beginning of a zbuf. If *zbufSeg* is `NULL`, *offset* is the absolute offset to any byte in the zbuf. However, it is more efficient to identify a zbuf byte location relative to the *zbufSeg* that contains it; see *zbufSegFind()* to convert any *zbufSeg*, *offset* pair to the most efficient equivalent.

Negative *offset* values always refer to bytes before the corresponding *zbufSeg*, and are usually not the most efficient address formulation in themselves (though using them may save your program other work in some cases).

The following special *offset* values, defined as constants, allow you to specify the very beginning or the very end of an entire zbuf, regardless of the *zbufSeg* value:

**ZBUF\_BEGIN**

The beginning of the entire zbuf.

**ZBUF\_END**

The end of the entire zbuf (useful for appending to a zbuf; see below).

**INSERTION AND LIMITS ON OFFSETS**

An *offset* is not valid if it points outside the zbuf. Thus, to address data currently within an N-byte zbuf, the valid offsets relative to the first segment are 0 through N-1.

Insertion routines are a special case: they obey the usual convention, but they use *offset* to specify where the new data begins after the insertion is complete. With regard to the original zbuf data, therefore, data is always inserted just before the byte location addressed by the *offset* value. The value of this convention is that it permits inserting (or concatenating) data either before or after the existing data. To insert before all the data currently in a zbuf segment, use 0 as *offset*. To insert after all the data in an N-byte segment, use N as *offset*. An *offset* of N-1 inserts the data just before the last byte in an N-byte segment.

An *offset* of 0 is always a valid insertion point; for an empty zbuf, 0 is the only valid *offset* (and `NULL` the only valid *zbufSeg*).

**SHARING DATA**

The routines in this library avoid copying segment data whenever possible. Thus, by passing and manipulating **ZBUF\_IDS** rather than copying data, multiple programs can communicate with greater efficiency. However, each program must be aware of data

sharing: changes to the data in a zbuf segment are visible to all zbuf segments that reference the data.

To alter your own program's view of zbuf data without affecting other programs, first use `zbufDup()` to make a new zbuf; then you can use an insertion or deletion routine, such as `zbufInsertBuf()`, to add a segment that only your program sees (until you pass a zbuf containing it to another program). It is safest to do all direct data manipulation in a private buffer, before enrolling it in a zbuf: in principle, you should regard all zbuf segment data as shared.

Once a data buffer is enrolled in a zbuf segment, the zbuf library is responsible for noticing when the buffer is no longer in use by any program, and freeing it. To support this, `zbufInsertBuf()` requires that you specify a callback to a free routine each time you build a zbuf segment around an existing buffer. You can use this callback to notify your application when a data buffer is no longer in use.

**SEE ALSO**      `zbufSockLib`, *VxWorks Programmer's Guide: Network*

---

## zbufSockLib

**NAME**            `zbufSockLib` – zbuf socket interface library

**ROUTINES**        `zbufSockLibInit()` – initialize the zbuf socket interface library  
`zbufSockSend()` – send zbuf data to a TCP socket  
`zbufSockSendto()` – send a zbuf message to a UDP socket  
`zbufSockBufSend()` – create a zbuf from user data and send it to a TCP socket  
`zbufSockBufSendto()` – create a zbuf from a user message and send it to a UDP socket  
`zbufSockRecv()` – receive data in a zbuf from a TCP socket  
`zbufSockRecvfrom()` – receive a message in a zbuf from a UDP socket

**DESCRIPTION**    This library contains routines that communicate over BSD sockets using the *zbuf interface* described in the `zbufLib` manual page. These zbuf socket calls communicate over BSD sockets in a similar manner to the socket routines in `sockLib`, but they avoid copying data unnecessarily between application buffers and network buffers.

**SEE ALSO**        `zbufLib`, `sockLib`, *VxWorks Programmer's Guide: Network*

# 2

## Subroutines

|                                   |                                                                      |      |
|-----------------------------------|----------------------------------------------------------------------|------|
| <i>a0()</i>                       | – return the contents of register a0 (also a1 – a7) (MC680x0) .....  | 2-1  |
| <i>abort()</i>                    | – cause abnormal program termination (ANSI) .....                    | 2-1  |
| <i>abs()</i>                      | – compute the absolute value of an integer (ANSI) .....              | 2-2  |
| <i>accept()</i>                   | – accept a connection from a socket .....                            | 2-2  |
| <i>acos()</i>                     | – compute an arc cosine (ANSI) .....                                 | 2-3  |
| <i>acosf()</i>                    | – compute an arc cosine (ANSI) .....                                 | 2-3  |
| <i>acw()</i>                      | – return the contents of the acw register (i960) .....               | 2-4  |
| <i>aic7880CtrlCreate()</i>        | – create a control structure for the AIC 7880 .....                  | 2-4  |
| <i>aic7880dFifoThresholdSet()</i> | – set the data FIFO threshold. ....                                  | 2-5  |
| <i>aic7880EnableFast20()</i>      | – enable double speed SCSI data transfers .....                      | 2-5  |
| <i>aic7880GetNumOfBuses()</i>     | – perform a PCI bus scan .....                                       | 2-6  |
| <i>aic7880ReadConfig()</i>        | – read from PCI config space .....                                   | 2-6  |
| <i>aic7880ScbCompleted()</i>      | – successfully completed execution of a client thread .....          | 2-7  |
| <i>aic7880WriteConfig()</i>       | – read to PCI config space .....                                     | 2-7  |
| <i>aioPxLibInit()</i>             | – initialize the asynchronous I/O (AIO) library .....                | 2-8  |
| <i>aioShow()</i>                  | – show AIO requests .....                                            | 2-8  |
| <i>aioSysInit()</i>               | – initialize the AIO system driver .....                             | 2-9  |
| <i>aio_error()</i>                | – retrieve error status of asynchronous I/O operation (POSIX) .....  | 2-9  |
| <i>aio_fsync()</i>                | – asynchronous file synchronization (POSIX) .....                    | 2-10 |
| <i>aio_read()</i>                 | – initiate an asynchronous read (POSIX) .....                        | 2-11 |
| <i>aio_return()</i>               | – retrieve return status of asynchronous I/O operation (POSIX) ..... | 2-11 |
| <i>aio_suspend()</i>              | – wait for asynchronous I/O request(s) (POSIX) .....                 | 2-12 |
| <i>aio_write()</i>                | – initiate an asynchronous write (POSIX) .....                       | 2-13 |
| <i>ambaDevInit()</i>              | – initialise an AMBA channel .....                                   | 2-13 |
| <i>ambaIntRx()</i>                | – handle a receiver interrupt .....                                  | 2-14 |
| <i>ambaIntTx()</i>                | – handle a transmitter interrupt .....                               | 2-14 |
| <i>arpAdd()</i>                   | – add an entry to the system ARP table .....                         | 2-15 |
| <i>arpDelete()</i>                | – delete an entry from the system ARP table .....                    | 2-16 |
| <i>arpFlush()</i>                 | – flush all entries in the system ARP table .....                    | 2-16 |
| <i>arpShow()</i>                  | – display entries in the system ARP table .....                      | 2-17 |
| <i>arptabShow()</i>               | – display the known ARP entries .....                                | 2-17 |

|                              |                                                                       |      |
|------------------------------|-----------------------------------------------------------------------|------|
| <i>asctime()</i>             | – convert broken-down time into a string (ANSI) .....                 | 2-18 |
| <i>asctime_r()</i>           | – convert broken-down time into a string (POSIX) .....                | 2-18 |
| <i>asin()</i>                | – compute an arc sine (ANSI) .....                                    | 2-19 |
| <i>asinf()</i>               | – compute an arc sine (ANSI) .....                                    | 2-19 |
| <i>assert()</i>              | – put diagnostics into programs (ANSI) .....                          | 2-20 |
| <i>ataDevCreate()</i>        | – create a device for a ATA/IDE disk .....                            | 2-20 |
| <i>ataDrv()</i>              | – initialize the ATA driver .....                                     | 2-21 |
| <i>atan()</i>                | – compute an arc tangent (ANSI) .....                                 | 2-22 |
| <i>atan2()</i>               | – compute the arc tangent of y/x (ANSI) .....                         | 2-22 |
| <i>atan2f()</i>              | – compute the arc tangent of y/x (ANSI) .....                         | 2-23 |
| <i>atanf()</i>               | – compute an arc tangent (ANSI) .....                                 | 2-23 |
| <i>ataRawio()</i>            | – do raw I/O access .....                                             | 2-24 |
| <i>ataShow()</i>             | – show the ATA/IDE disk parameters .....                              | 2-24 |
| <i>ataShowInit()</i>         | – initialize the ATA/IDE disk driver show routine .....               | 2-25 |
| <i>atexit()</i>              | – call a function at program termination (Unimplemented) (ANSI) ..... | 2-25 |
| <i>atof()</i>                | – convert a string to a double (ANSI) .....                           | 2-26 |
| <i>atoi()</i>                | – convert a string to an int (ANSI) .....                             | 2-26 |
| <i>atol()</i>                | – convert a string to a long (ANSI) .....                             | 2-27 |
| <i>b()</i>                   | – set or display breakpoints .....                                    | 2-27 |
| <i>bcmp()</i>                | – compare one buffer to another .....                                 | 2-28 |
| <i>bcopy()</i>               | – copy one buffer to another .....                                    | 2-29 |
| <i>bcopyBytes()</i>          | – copy one buffer to another one byte at a time .....                 | 2-29 |
| <i>bcopyDoubles()</i>        | – copy one buffer to another eight bytes at a time (SPARC) .....      | 2-30 |
| <i>bcopyLongs()</i>          | – copy one buffer to another one long word at a time .....            | 2-30 |
| <i>bcopyWords()</i>          | – copy one buffer to another one word at a time .....                 | 2-31 |
| <i>bd()</i>                  | – delete a breakpoint .....                                           | 2-31 |
| <i>bdall()</i>               | – delete all breakpoints .....                                        | 2-32 |
| <i>bfill()</i>               | – fill a buffer with a specified character .....                      | 2-32 |
| <i>bfillBytes()</i>          | – fill buffer with a specified character one byte at a time .....     | 2-33 |
| <i>bfillDoubles()</i>        | – fill a buffer with a specified eight-byte pattern (SPARC) .....     | 2-33 |
| <i>bh()</i>                  | – set a hardware breakpoint .....                                     | 2-34 |
| <i>bind()</i>                | – bind a name to a socket .....                                       | 2-34 |
| <i>bindresvport()</i>        | – bind a socket to a privileged IP port .....                         | 2-35 |
| <i>binvert()</i>             | – invert the order of bytes in a buffer .....                         | 2-35 |
| <i>bootBpAnchorExtract()</i> | – extract a backplane address from a device field .....               | 2-36 |
| <i>bootChange()</i>          | – change the boot line .....                                          | 2-36 |
| <i>bootLeaseExtract()</i>    | – extract the lease information from an Internet address .....        | 2-37 |
| <i>bootNetmaskExtract()</i>  | – extract the net mask field from an Internet address .....           | 2-38 |
| <i>bootParamsPrompt()</i>    | – prompt for boot line parameters .....                               | 2-39 |
| <i>bootParamsShow()</i>      | – display boot line parameters .....                                  | 2-39 |
| <i>bootpMsgSend()</i>        | – send a BOOTP request message .....                                  | 2-40 |
| <i>bootpParamsGet()</i>      | – retrieve boot parameters using BOOTP .....                          | 2-41 |
| <i>bootStringToStruct()</i>  | – interpret the boot parameters from the boot line .....              | 2-44 |
| <i>bootStructToString()</i>  | – construct a boot line .....                                         | 2-44 |
| <i>bsearch()</i>             | – perform a binary search (ANSI) .....                                | 2-45 |
| <i>bswap()</i>               | – swap buffers .....                                                  | 2-46 |
| <i>bzero()</i>               | – zero out a buffer .....                                             | 2-46 |
| <i>bzeroDoubles()</i>        | – zero out a buffer eight bytes at a time (SPARC) .....               | 2-47 |



|                                              |                                                                      |      |
|----------------------------------------------|----------------------------------------------------------------------|------|
| <code>c()</code>                             | – continue from a breakpoint .....                                   | 2-47 |
| <code>cacheArchClearEntry()</code>           | – clear an entry from a cache (68K, x86) .....                       | 2-48 |
| <code>cacheArchLibInit()</code>              | – initialize the cache library .....                                 | 2-48 |
| <code>cacheClear()</code>                    | – clear all or some entries from a cache .....                       | 2-50 |
| <code>cacheCy604ClearLine()</code>           | – clear a line from a CY7C604 cache .....                            | 2-51 |
| <code>cacheCy604ClearPage()</code>           | – clear a page from a CY7C604 cache .....                            | 2-51 |
| <code>cacheCy604ClearRegion()</code>         | – clear a region from a CY7C604 cache .....                          | 2-52 |
| <code>cacheCy604ClearSegment()</code>        | – clear a segment from a CY7C604 cache .....                         | 2-52 |
| <code>cacheCy604LibInit()</code>             | – initialize the Cypress CY7C604 cache library .....                 | 2-53 |
| <code>cacheDisable()</code>                  | – disable the specified cache .....                                  | 2-53 |
| <code>cacheDmaFree()</code>                  | – free the buffer acquired with <code>cacheDmaMalloc()</code> .....  | 2-54 |
| <code>cacheDmaMalloc()</code>                | – allocate a cache-safe buffer for DMA devices and drivers .....     | 2-54 |
| <code>cacheDrvFlush()</code>                 | – flush the data cache for drivers .....                             | 2-55 |
| <code>cacheDrvInvalidate()</code>            | – invalidate data cache for drivers .....                            | 2-55 |
| <code>cacheDrvPhysToVirt()</code>            | – translate a physical address for drivers .....                     | 2-56 |
| <code>cacheDrvVirtToPhys()</code>            | – translate a virtual address for drivers .....                      | 2-56 |
| <code>cacheEnable()</code>                   | – enable the specified cache .....                                   | 2-57 |
| <code>cacheFlush()</code>                    | – flush all or some of a specified cache .....                       | 2-57 |
| <code>cacheI960CxIC1kLoadNLock()</code>      | – load and lock I960Cx 1KB instruction cache (i960) .....            | 2-58 |
| <code>cacheI960CxICDisable()</code>          | – disable the I960Cx instruction cache (i960) .....                  | 2-58 |
| <code>cacheI960CxICEnable()</code>           | – enable the I960Cx instruction cache (i960) .....                   | 2-58 |
| <code>cacheI960CxICInvalidate()</code>       | – invalidate the I960Cx instruction cache (i960) .....               | 2-59 |
| <code>cacheI960CxICLoadNLock()</code>        | – load and lock I960Cx 512-byte instruction cache (i960) .....       | 2-59 |
| <code>cacheI960CxLibInit()</code>            | – initialize the I960Cx cache library (i960) .....                   | 2-60 |
| <code>cacheI960JxDCCoherent()</code>         | – ensure data cache coherency (i960) .....                           | 2-60 |
| <code>cacheI960JxDCDisable()</code>          | – disable the I960Jx data cache (i960) .....                         | 2-60 |
| <code>cacheI960JxDCEnable()</code>           | – enable the I960Jx data cache (i960) .....                          | 2-61 |
| <code>cacheI960JxDCFlush()</code>            | – flush the I960Jx data cache (i960) .....                           | 2-61 |
| <code>cacheI960JxDCInvalidate()</code>       | – invalidate the I960Jx data cache (i960) .....                      | 2-61 |
| <code>cacheI960JxDCStatusGet()</code>        | – get the I960Jx data cache status (i960) .....                      | 2-62 |
| <code>cacheI960JxICDisable()</code>          | – disable the I960Jx instruction cache (i960) .....                  | 2-62 |
| <code>cacheI960JxICEnable()</code>           | – enable the I960Jx instruction cache (i960) .....                   | 2-62 |
| <code>cacheI960JxICFlush()</code>            | – flush the I960Jx instruction cache (i960) .....                    | 2-63 |
| <code>cacheI960JxICInvalidate()</code>       | – invalidate the I960Jx instruction cache (i960) .....               | 2-63 |
| <code>cacheI960JxICLoadNLock()</code>        | – load and lock the I960Jx instruction cache (i960) .....            | 2-63 |
| <code>cacheI960JxICLockingStatusGet()</code> | – get the I960Jx I-cache locking status (i960) .....                 | 2-64 |
| <code>cacheI960JxICStatusGet()</code>        | – get the I960Jx instruction cache status (i960) .....               | 2-64 |
| <code>cacheI960JxLibInit()</code>            | – initialize the I960Jx cache library (i960) .....                   | 2-64 |
| <code>cacheInvalidate()</code>               | – invalidate all or some of a specified cache .....                  | 2-65 |
| <code>cacheLibInit()</code>                  | – initialize the cache library for a processor architecture .....    | 2-65 |
| <code>cacheLock()</code>                     | – lock all or part of a specified cache .....                        | 2-66 |
| <code>cacheMb930ClearLine()</code>           | – clear a line from an MB86930 cache .....                           | 2-66 |
| <code>cacheMb930LibInit()</code>             | – initialize the Fujitsu MB86930 cache library .....                 | 2-67 |
| <code>cacheMb930LockAuto()</code>            | – enable MB86930 automatic locking of kernel instructions/data ..... | 2-67 |
| <code>cacheMicroSparcLibInit()</code>        | – initialize the microSPARC cache library .....                      | 2-68 |
| <code>cachePipeFlush()</code>                | – flush processor write buffers to memory .....                      | 2-68 |
| <code>cacheR3kDsize()</code>                 | – return the size of the R3000 data cache .....                      | 2-69 |
| <code>cacheR3kIsize()</code>                 | – return the size of the R3000 instruction cache .....               | 2-69 |

|                                  |                                                                                        |      |
|----------------------------------|----------------------------------------------------------------------------------------|------|
| <i>cacheR3kLibInit()</i>         | – initialize the R3000 cache library .....                                             | 2-69 |
| <i>cacheR4kLibInit()</i>         | – initialize the R4000 cache library .....                                             | 2-70 |
| <i>cacheR33kLibInit()</i>        | – initialize the R33000 cache library .....                                            | 2-70 |
| <i>cacheR333x0LibInit()</i>      | – initialize the R333x0 cache library .....                                            | 2-71 |
| <i>cacheStoreBufDisable()</i>    | – disable the store buffer (MC68060 only) .....                                        | 2-71 |
| <i>cacheStoreBufEnable()</i>     | – enable the store buffer (MC68060 only) .....                                         | 2-71 |
| <i>cacheSun4ClearContext()</i>   | – clear a specific context from a Sun-4 cache .....                                    | 2-72 |
| <i>cacheSun4ClearLine()</i>      | – clear a line from a Sun-4 cache .....                                                | 2-72 |
| <i>cacheSun4ClearPage()</i>      | – clear a page from a Sun-4 cache .....                                                | 2-73 |
| <i>cacheSun4ClearSegment()</i>   | – clear a segment from a Sun-4 cache .....                                             | 2-73 |
| <i>cacheSun4LibInit()</i>        | – initialize the Sun-4 cache library .....                                             | 2-74 |
| <i>cacheTextUpdate()</i>         | – synchronize the instruction and data caches .....                                    | 2-74 |
| <i>cacheTiTms390LibInit()</i>    | – initialize the TI TMS390 cache library .....                                         | 2-75 |
| <i>cacheTiTms390PhysToVirt()</i> | – translate a physical address for drivers .....                                       | 2-75 |
| <i>cacheTiTms390VirtToPhys()</i> | – translate a virtual address for cacheLib .....                                       | 2-76 |
| <i>cacheUnlock()</i>             | – unlock all or part of a specified cache .....                                        | 2-76 |
| <i>calloc()</i>                  | – allocate space for an array (ANSI) .....                                             | 2-77 |
| <i>cbrt()</i>                    | – compute a cube root .....                                                            | 2-77 |
| <i>cbrtf()</i>                   | – compute a cube root .....                                                            | 2-78 |
| <i>cd()</i>                      | – change the default directory .....                                                   | 2-78 |
| <i>cd2400HrdInit()</i>           | – initialize the chip .....                                                            | 2-79 |
| <i>cd2400Int()</i>               | – handle special status interrupts .....                                               | 2-80 |
| <i>cd2400IntRx()</i>             | – handle receiver interrupts .....                                                     | 2-80 |
| <i>cd2400IntTx()</i>             | – handle transmitter interrupts .....                                                  | 2-80 |
| <i>cdromFsDevCreate()</i>        | – create a cdromFsLib device .....                                                     | 2-81 |
| <i>cdromFsInit()</i>             | – initialize cdromFsLib .....                                                          | 2-81 |
| <i>cdromFsVolConfigShow()</i>    | – show the volume configuration information .....                                      | 2-82 |
| <i>ceil()</i>                    | – compute the smallest integer greater than or equal to a specified value (ANSI) ..... | 2-82 |
| <i>ceilf()</i>                   | – compute the smallest integer greater than or equal to a specified value (ANSI) ..... | 2-83 |
| <i>cfree()</i>                   | – free a block of memory .....                                                         | 2-83 |
| <i>chdir()</i>                   | – set the current default path .....                                                   | 2-84 |
| <i>checkStack()</i>              | – print a summary of each task’s stack usage .....                                     | 2-84 |
| <i>cisConfigregGet()</i>         | – get the PCMCIA configuration register .....                                          | 2-85 |
| <i>cisConfigregSet()</i>         | – set the PCMCIA configuration register .....                                          | 2-85 |
| <i>cisFree()</i>                 | – free tuples from the linked list .....                                               | 2-86 |
| <i>cisGet()</i>                  | – get information from a PC card’s CIS .....                                           | 2-86 |
| <i>cisShow()</i>                 | – show CIS information .....                                                           | 2-87 |
| <i>cleanupStoreBuffer()</i>      | – clean up store buffer after a data store error interrupt .....                       | 2-87 |
| <i>clearerr()</i>                | – clear end-of-file and error flags for a stream (ANSI) .....                          | 2-88 |
| <i>clock()</i>                   | – determine the processor time in use (ANSI) .....                                     | 2-88 |
| <i>clock_getres()</i>            | – get the clock resolution (POSIX) .....                                               | 2-89 |
| <i>clock_gettime()</i>           | – get the current time of the clock (POSIX) .....                                      | 2-89 |
| <i>clock_setres()</i>            | – set the clock resolution .....                                                       | 2-90 |
| <i>clock_settime()</i>           | – set the clock to a specified time (POSIX) .....                                      | 2-90 |
| <i>close()</i>                   | – close a file .....                                                                   | 2-91 |
| <i>closedir()</i>                | – close a directory (POSIX) .....                                                      | 2-91 |
| <i>connect()</i>                 | – initiate a connection to a socket .....                                              | 2-92 |
| <i>connectWithTimeout()</i>      | – try to connect over a socket for a specified duration .....                          | 2-92 |

|                               |                                                                            |       |
|-------------------------------|----------------------------------------------------------------------------|-------|
| <i>copy()</i>                 | – copy in (or stdin) to out (or stdout) .....                              | 2-93  |
| <i>copyStreams()</i>          | – copy from/to specified streams .....                                     | 2-94  |
| <i>cos()</i>                  | – compute a cosine (ANSI) .....                                            | 2-94  |
| <i>cosf()</i>                 | – compute a cosine (ANSI) .....                                            | 2-95  |
| <i>cosh()</i>                 | – compute a hyperbolic cosine (ANSI) .....                                 | 2-95  |
| <i>coshf()</i>                | – compute a hyperbolic cosine (ANSI) .....                                 | 2-96  |
| <i>cplusCallNewHandler()</i>  | – call the allocation failure handler (C++) .....                          | 2-96  |
| <i>cplusCtors()</i>           | – call static constructors (C++) .....                                     | 2-97  |
| <i>cplusCtorsLink()</i>       | – call all linked static constructors (C++) .....                          | 2-97  |
| <i>cplusDemanglerSet()</i>    | – change C++ demangling mode (C++) .....                                   | 2-98  |
| <i>cplusDtors()</i>           | – call static destructors (C++) .....                                      | 2-99  |
| <i>cplusDtorsLink()</i>       | – call all linked static destructors (C++) .....                           | 2-99  |
| <i>cplusLibInit()</i>         | – initialize the C++ library (C++) .....                                   | 2-100 |
| <i>cplusXtorSet()</i>         | – change C++ static constructor calling strategy (C++) .....               | 2-100 |
| <i>cpmattach()</i>            | – publish the cpm network interface and initialize the driver .....        | 2-101 |
| <i>cpmStartOutput()</i>       | – output packet to network interface device .....                          | 2-102 |
| <i>cpsr()</i>                 | – return the contents of the current processor status register (ARM) ..... | 2-103 |
| <i>creat()</i>                | – create a file .....                                                      | 2-104 |
| <i>cret()</i>                 | – continue until the current subroutine returns .....                      | 2-104 |
| <i>csAttach()</i>             | – publish the cs network interface and initialize the driver. ....         | 2-105 |
| <i>csShow()</i>               | – shows statistics for the cs network interface .....                      | 2-106 |
| <i>ctime()</i>                | – convert time in seconds into a string (ANSI) .....                       | 2-106 |
| <i>ctime_r()</i>              | – convert time in seconds into a string (POSIX) .....                      | 2-107 |
| <i>d()</i>                    | – display memory .....                                                     | 2-107 |
| <i>d0()</i>                   | – return the contents of register d0 (also d1 – d7) (MC680x0) .....        | 2-108 |
| <i>dbgBpTypeBind()</i>        | – bind a breakpoint handler to a breakpoint type (MIPS R3000, R4000) ..... | 2-108 |
| <i>dbgHelp()</i>              | – display debugging help menu .....                                        | 2-109 |
| <i>dbgInit()</i>              | – initialize the local debugging package .....                             | 2-110 |
| <i>dcattach()</i>             | – publish the dc network interface. ....                                   | 2-110 |
| <i>dcCsrShow()</i>            | – display dec 21040/21140 status registers 0 thru 15 .....                 | 2-111 |
| <i>dcReadAllRom()</i>         | – read entire serial rom .....                                             | 2-112 |
| <i>dcViewRom()</i>            | – display lines of serial ROM for dec21140 .....                           | 2-112 |
| <i>dec21x4xEndLoad()</i>      | – initialize the driver and device .....                                   | 2-113 |
| <i>dec21x40EndLoad()</i>      | – initialize the driver and device .....                                   | 2-113 |
| <i>dec21x40PhyLinkPoll()</i>  | – Poll the PHY for link status .....                                       | 2-114 |
| <i>dec21140SromWordRead()</i> | – read two bytes from the serial ROM .....                                 | 2-114 |
| <i>devs()</i>                 | – list all system-known devices .....                                      | 2-115 |
| <i>dhcpcBind()</i>            | – obtain a set of network configuration parameters with DHCP .....         | 2-115 |
| <i>dhcpcBootBind()</i>        | – initialize the network with DHCP at boot time .....                      | 2-116 |
| <i>dhcpcBootInit()</i>        | – set up the DHCP client parameters and data structures .....              | 2-117 |
| <i>dhcpcBootOptionSet()</i>   | – add an option to the option request list .....                           | 2-117 |
| <i>dhcpcCacheHookAdd()</i>    | – add a routine to store and retrieve lease data .....                     | 2-119 |
| <i>dhcpcCacheHookDelete()</i> | – delete a lease data storage routine .....                                | 2-120 |
| <i>dhcpcEventHookAdd()</i>    | – add a routine to handle configuration parameters .....                   | 2-121 |
| <i>dhcpcEventHookDelete()</i> | – remove the configuration parameters handler .....                        | 2-122 |
| <i>dhcpcInit()</i>            | – assign network interface and setup lease request .....                   | 2-122 |
| <i>dhcpcLibInit()</i>         | – DHCP client library initialization .....                                 | 2-123 |
| <i>dhcpcOptionGet()</i>       | – retrieve an option provided to a client and store in a buffer .....      | 2-124 |

|                                 |                                                                                |       |
|---------------------------------|--------------------------------------------------------------------------------|-------|
| <i>dhcpcOptionSet()</i>         | – add an option to the option request list .....                               | 2-125 |
| <i>dhcpcParamsGet()</i>         | – retrieve current configuration parameters .....                              | 2-127 |
| <i>dhcpcParamsShow()</i>        | – display current lease parameters .....                                       | 2-128 |
| <i>dhcpcRelease()</i>           | – relinquish specified lease .....                                             | 2-128 |
| <i>dhcpcServerGet()</i>         | – retrieve the current DHCP server .....                                       | 2-129 |
| <i>dhcpcServerShow()</i>        | – display current DHCP server .....                                            | 2-129 |
| <i>dhcpcShowInit()</i>          | – initialize the DHCP show facility .....                                      | 2-130 |
| <i>dhcpcShutdown()</i>          | – disable DHCP client library .....                                            | 2-130 |
| <i>dhcpcTimerGet()</i>          | – retrieve current lease timers .....                                          | 2-131 |
| <i>dhcpcTimersShow()</i>        | – display current lease timers .....                                           | 2-131 |
| <i>dhcpcVerify()</i>            | – renew an established lease .....                                             | 2-132 |
| <i>dhcpsAddressHookAdd()</i>    | – assign a permanent address storage hook for the server .....                 | 2-132 |
| <i>dhcpsInit()</i>              | – set up the DHCP server parameters and data structures .....                  | 2-133 |
| <i>dhcpsLeaseEntryAdd()</i>     | – add another entry to the address pool .....                                  | 2-134 |
| <i>dhcpsLeaseHookAdd()</i>      | – assign a permanent lease storage hook for the server .....                   | 2-135 |
| <i>difftime()</i>               | – compute the difference between two calendar times (ANSI) .....               | 2-136 |
| <i>diskFormat()</i>             | – format a disk .....                                                          | 2-136 |
| <i>diskInit()</i>               | – initialize a file system on a block device .....                             | 2-137 |
| <i>div()</i>                    | – compute a quotient and remainder (ANSI) .....                                | 2-137 |
| <i>div_r()</i>                  | – compute a quotient and remainder (reentrant) .....                           | 2-138 |
| <i>dosFsConfigGet()</i>         | – obtain dosFs volume configuration values .....                               | 2-138 |
| <i>dosFsConfigInit()</i>        | – initialize dosFs volume configuration structure .....                        | 2-139 |
| <i>dosFsConfigShow()</i>        | – display dosFs volume configuration data .....                                | 2-140 |
| <i>dosFsDateSet()</i>           | – set the dosFs file system date .....                                         | 2-140 |
| <i>dosFsDateTimeInstall()</i>   | – install a user-supplied date/time function .....                             | 2-141 |
| <i>dosFsDevInit()</i>           | – associate a block device with dosFs file system functions .....              | 2-141 |
| <i>dosFsDevInitOptionsSet()</i> | – specify volume options for <i>dosFsDevInit()</i> .....                       | 2-142 |
| <i>dosFsInit()</i>              | – prepare to use the dosFs library .....                                       | 2-143 |
| <i>dosFsMkfs()</i>              | – initialize a device and create a dosFs file system .....                     | 2-144 |
| <i>dosFsMkfsOptionsSet()</i>    | – specify volume options for <i>dosFsMkfs()</i> .....                          | 2-145 |
| <i>dosFsModeChange()</i>        | – modify the mode of a dosFs volume .....                                      | 2-145 |
| <i>dosFsReadyChange()</i>       | – notify dosFs of a change in ready status .....                               | 2-146 |
| <i>dosFsTimeSet()</i>           | – set the dosFs file system time .....                                         | 2-146 |
| <i>dosFsVolOptionsGet()</i>     | – get current dosFs volume options .....                                       | 2-147 |
| <i>dosFsVolOptionsSet()</i>     | – set dosFs volume options .....                                               | 2-147 |
| <i>dosFsVolUnmount()</i>        | – unmount a dosFs volume .....                                                 | 2-148 |
| <i>dummyCallback()</i>          | – dummy callback routine .....                                                 | 2-149 |
| <i>dummyCallback()</i>          | – dummy callback routine. ....                                                 | 2-149 |
| <i>e()</i>                      | – set or display eventpoints (WindView) .....                                  | 2-149 |
| <i>edi()</i>                    | – return the contents of register edi (also esi – eax) (i386/i486) .....       | 2-150 |
| <i>eexattach()</i>              | – publish the eex network interface and initialize the driver and device ..... | 2-151 |
| <i>eexTxStartup()</i>           | – start output on the chip .....                                               | 2-151 |
| <i>eflags()</i>                 | – return the contents of the status register (i386/i486) .....                 | 2-152 |
| <i>ei82596EndLoad()</i>         | – initialize the driver and device .....                                       | 2-153 |
| <i>eiattach()</i>               | – publish the ei network interface and initialize the driver and device .....  | 2-154 |
| <i>eihkattach()</i>             | – publish the ei network interface and initialize the driver and device .....  | 2-155 |
| <i>eiInt()</i>                  | – entry point for handling interrupts from the 82596 .....                     | 2-156 |
| <i>eiTxStartup()</i>            | – start output on the chip .....                                               | 2-156 |

|                                |                                                                                |       |
|--------------------------------|--------------------------------------------------------------------------------|-------|
| <i>eiTxStartup()</i>           | – start output on the chip .....                                               | 2-157 |
| <i>el3c90xEndLoad()</i>        | – initialize the driver and device .....                                       | 2-158 |
| <i>el3c90xInitParse()</i>      | – parse the initialization string .....                                        | 2-159 |
| <i>elcattach()</i>             | – publish the elc network interface and initialize the driver and device ..... | 2-160 |
| <i>elcPut()</i>                | – copy a packet to the interface. ....                                         | 2-160 |
| <i>elcShow()</i>               | – display statistics for the SMC 8013WC elc network interface .....            | 2-161 |
| <i>elt3c509Load()</i>          | – initialize the driver and device .....                                       | 2-161 |
| <i>elt3c509Parse()</i>         | – parse the init string .....                                                  | 2-162 |
| <i>eltattach()</i>             | – publish the elt interface and initialize the driver and device .....         | 2-163 |
| <i>eltShow()</i>               | – display statistics for the 3C509 elt network interface .....                 | 2-163 |
| <i>eltTxOutputStart()</i>      | – start output on the board .....                                              | 2-164 |
| <i>endEtherAddressForm()</i>   | – form an Ethernet address into a packet .....                                 | 2-164 |
| <i>endEtherPacketAddrGet()</i> | – locate the addresses in a packet .....                                       | 2-165 |
| <i>endEtherPacketDataGet()</i> | – return the beginning of the packet data .....                                | 2-165 |
| <i>endFindByName()</i>         | – find a device using its string name .....                                    | 2-166 |
| <i>endObjFlagSet()</i>         | – set the flags member of an END_OBJ structure .....                           | 2-166 |
| <i>endObjInit()</i>            | – initialize an END_OBJ structure .....                                        | 2-167 |
| <i>eneattach()</i>             | – publish the ene network interface and initialize the driver and device ..... | 2-167 |
| <i>enePut()</i>                | – copy a packet to the interface. ....                                         | 2-168 |
| <i>eneShow()</i>               | – display statistics for the NE2000 ene network interface .....                | 2-168 |
| <i>envLibInit()</i>            | – initialize environment variable facility .....                               | 2-169 |
| <i>envoy_call_timer()</i>      | – execute the specified function when the timer expires .....                  | 2-169 |
| <i>envoy_now()</i>             | – return the number of clock ticks elapsed since the timer was set .....       | 2-170 |
| <i>envPrivateCreate()</i>      | – create a private environment .....                                           | 2-170 |
| <i>envPrivateDestroy()</i>     | – destroy a private environment .....                                          | 2-171 |
| <i>envShow()</i>               | – display the environment for a task .....                                     | 2-171 |
| <i>errnoGet()</i>              | – get the error status value of the calling task .....                         | 2-172 |
| <i>errnoOfTaskGet()</i>        | – get the error status value of a specified task .....                         | 2-172 |
| <i>errnoOfTaskSet()</i>        | – set the error status value of a specified task .....                         | 2-173 |
| <i>errnoSet()</i>              | – set the error status value of the calling task .....                         | 2-173 |
| <i>esmcattach()</i>            | – publish the esmc network interface and initialize the driver. ....           | 2-174 |
| <i>esmcPut()</i>               | – copy a packet to the interface. ....                                         | 2-174 |
| <i>esmcShow()</i>              | – display statistics for the esmc network interface .....                      | 2-175 |
| <i>etherAddrResolve()</i>      | – resolve an Ethernet address for a specified Internet address .....           | 2-175 |
| <i>etherInputHookAdd()</i>     | – add a routine to receive all Ethernet input packets .....                    | 2-176 |
| <i>etherInputHookDelete()</i>  | – delete a network interface input hook routine .....                          | 2-177 |
| <i>etherMultiAdd()</i>         | – add multicast address to a multicast address list .....                      | 2-178 |
| <i>etherMultiDel()</i>         | – delete an Ethernet multicast address record .....                            | 2-178 |
| <i>etherMultiGet()</i>         | – retrieve a table of multicast addresses from a driver .....                  | 2-179 |
| <i>etherOutput()</i>           | – send a packet on an Ethernet interface .....                                 | 2-179 |
| <i>etherOutputHookAdd()</i>    | – add a routine to receive all Ethernet output packets .....                   | 2-180 |
| <i>etherOutputHookDelete()</i> | – delete a network interface output hook routine .....                         | 2-181 |
| <i>etherTypeGet()</i>          | – get the type from an ethernet packet .....                                   | 2-182 |
| <i>evbNs16550HrdInit()</i>     | – initialize the NS 16550 chip .....                                           | 2-182 |
| <i>evbNs16550Int()</i>         | – handle a receiver/transmitter interrupt for the NS 16550 chip .....          | 2-183 |
| <i>excConnect()</i>            | – connect a C routine to an exception vector (PowerPC) .....                   | 2-183 |
| <i>excCrtConnect()</i>         | – connect a C routine to a critical exception vector (PowerPC 403) .....       | 2-184 |
| <i>excHookAdd()</i>            | – specify a routine to be called with exceptions .....                         | 2-185 |

|                           |                                                                                    |       |
|---------------------------|------------------------------------------------------------------------------------|-------|
| <i>excInit()</i>          | – initialize the exception handling package .....                                  | 2-185 |
| <i>excIntConnect()</i>    | – connect a C routine to an asynchronous exception vector (PowerPC, ARM) ....      | 2-186 |
| <i>excIntCrtConnect()</i> | – connect a C routine to a critical interrupt vector (PowerPC 403) .....           | 2-187 |
| <i>excTask()</i>          | – handle task-level exceptions .....                                               | 2-188 |
| <i>excVecGet()</i>        | – get a CPU exception vector (PowerPC, ARM) .....                                  | 2-188 |
| <i>excVecInit()</i>       | – initialize the exception/interrupt vectors .....                                 | 2-188 |
| <i>excVecSet()</i>        | – set a CPU exception vector (PowerPC, ARM) .....                                  | 2-189 |
| <i>exit()</i>             | – exit a task (ANSI) .....                                                         | 2-190 |
| <i>exp()</i>              | – compute an exponential value (ANSI) .....                                        | 2-190 |
| <i>expf()</i>             | – compute an exponential value (ANSI) .....                                        | 2-191 |
| <i>fabs()</i>             | – compute an absolute value (ANSI) .....                                           | 2-191 |
| <i>fabsf()</i>            | – compute an absolute value (ANSI) .....                                           | 2-192 |
| <i>fclose()</i>           | – close a stream (ANSI) .....                                                      | 2-192 |
| <i>fdDevCreate()</i>      | – create a device for a floppy disk .....                                          | 2-193 |
| <i>fdDrv()</i>            | – initialize the floppy disk driver .....                                          | 2-194 |
| <i>fdopen()</i>           | – open a file specified by a file descriptor (POSIX) .....                         | 2-195 |
| <i>fdprintf()</i>         | – write a formatted string to a file descriptor .....                              | 2-195 |
| <i>fdRawIot()</i>         | – provide raw I/O access .....                                                     | 2-196 |
| <i>fei82557EndLoad()</i>  | – initialize the driver and device .....                                           | 2-196 |
| <i>feiattach()</i>        | – publish the fei network interface .....                                          | 2-197 |
| <i>feof()</i>             | – test the end-of-file indicator for a stream (ANSI) .....                         | 2-198 |
| <i>ferror()</i>           | – test the error indicator for a file pointer (ANSI) .....                         | 2-199 |
| <i>fflush()</i>           | – flush a stream (ANSI) .....                                                      | 2-199 |
| <i>fgetc()</i>            | – return the next character from a stream (ANSI) .....                             | 2-200 |
| <i>fgetpos()</i>          | – store the current value of the file position indicator for a stream (ANSI) ..... | 2-200 |
| <i>fgets()</i>            | – read a specified number of characters from a stream (ANSI) .....                 | 2-201 |
| <i>fileno()</i>           | – return the file descriptor for a stream (POSIX) .....                            | 2-201 |
| <i>fioFormatV()</i>       | – convert a format string .....                                                    | 2-202 |
| <i>fioLibInit()</i>       | – initialize the formatted I/O support library .....                               | 2-203 |
| <i>fioRdString()</i>      | – read a string from a file .....                                                  | 2-203 |
| <i>fioRead()</i>          | – read a buffer .....                                                              | 2-204 |
| <i>floatInit()</i>        | – initialize floating-point I/O support .....                                      | 2-204 |
| <i>floor()</i>            | – compute the largest integer less than or equal to a specified value (ANSI) ..... | 2-205 |
| <i>floorf()</i>           | – compute the largest integer less than or equal to a specified value (ANSI) ..... | 2-205 |
| <i>fmod()</i>             | – compute the remainder of x/y (ANSI) .....                                        | 2-206 |
| <i>fmodf()</i>            | – compute the remainder of x/y (ANSI) .....                                        | 2-206 |
| <i>fnattach()</i>         | – publish the fn network interface and initialize the driver and device .....      | 2-207 |
| <i>fopen()</i>            | – open a file specified by name (ANSI) .....                                       | 2-207 |
| <i>fp()</i>               | – return the contents of register fp (i960) .....                                  | 2-209 |
| <i>fp0()</i>              | – return the contents of register fp0 (also fp1 – fp3) (i960KB, i960SB) .....      | 2-209 |
| <i>fppInit()</i>          | – initialize floating-point coprocessor support .....                              | 2-210 |
| <i>fppProbe()</i>         | – probe for the presence of a floating-point coprocessor .....                     | 2-210 |
| <i>fppRestore()</i>       | – restore the floating-point coprocessor context .....                             | 2-211 |
| <i>fppSave()</i>          | – save the floating-point coprocessor context .....                                | 2-212 |
| <i>fppShowInit()</i>      | – initialize the floating-point show facility .....                                | 2-213 |
| <i>fppTaskRegsGet()</i>   | – get the floating-point registers from a task TCB .....                           | 2-213 |
| <i>fppTaskRegsSet()</i>   | – set the floating-point registers of a task .....                                 | 2-214 |
| <i>fppTaskRegsShow()</i>  | – print the contents of a task’s floating-point registers .....                    | 2-214 |

|                          |                                                                                      |       |
|--------------------------|--------------------------------------------------------------------------------------|-------|
| <i>fprintf()</i>         | – write a formatted string to a stream (ANSI) .....                                  | 2-215 |
| <i>fputc()</i>           | – write a character to a stream (ANSI) .....                                         | 2-219 |
| <i>fputs()</i>           | – write a string to a stream (ANSI) .....                                            | 2-219 |
| <i>fread()</i>           | – read data into an array (ANSI) .....                                               | 2-220 |
| <i>free()</i>            | – free a block of memory (ANSI) .....                                                | 2-220 |
| <i>freopen()</i>         | – open a file specified by name (ANSI) .....                                         | 2-221 |
| <i>frexp()</i>           | – break floating-point number into normalized fraction and power of 2 (ANSI) .....   | 2-221 |
| <i>fscanf()</i>          | – read and convert characters from a stream (ANSI) .....                             | 2-222 |
| <i>fseek()</i>           | – set the file position indicator for a stream (ANSI) .....                          | 2-226 |
| <i>fsetpos()</i>         | – set the file position indicator for a stream (ANSI) .....                          | 2-227 |
| <i>fsrShow()</i>         | – display the meaning of a specified fsr value, symbolically (SPARC) .....           | 2-227 |
| <i>fstat()</i>           | – get file status information (POSIX) .....                                          | 2-228 |
| <i>fstatfs()</i>         | – get file status information (POSIX) .....                                          | 2-229 |
| <i>ftell()</i>           | – return the current value of the file position indicator for a stream (ANSI) .....  | 2-229 |
| <i>ftpCommand()</i>      | – send an FTP command and get the reply .....                                        | 2-230 |
| <i>ftpDataConnGet()</i>  | – get a completed FTP data connection .....                                          | 2-231 |
| <i>ftpDataConnInit()</i> | – initialize an FTP data connection .....                                            | 2-231 |
| <i>ftpdDelete()</i>      | – terminate the FTP server task .....                                                | 2-232 |
| <i>ftpdInit()</i>        | – initialize the FTP server task .....                                               | 2-232 |
| <i>ftpHookup()</i>       | – get a control connection to the FTP server on a specified host .....               | 2-233 |
| <i>ftpLogin()</i>        | – log in to a remote FTP server .....                                                | 2-234 |
| <i>ftpLs()</i>           | – list directory contents via FTP .....                                              | 2-234 |
| <i>ftpReplyGet()</i>     | – get an FTP command reply .....                                                     | 2-235 |
| <i>ftpXfer()</i>         | – initiate a transfer via FTP .....                                                  | 2-235 |
| <i>ftruncate()</i>       | – truncate a file (POSIX) .....                                                      | 2-237 |
| <i>fwrite()</i>          | – write from a specified array (ANSI) .....                                          | 2-238 |
| <i>g0()</i>              | – return the contents of register g0, also g1 – g7 (SPARC) and g1 – g14 (i960) ..... | 2-238 |
| <i>getc()</i>            | – return the next character from a stream (ANSI) .....                               | 2-239 |
| <i>getchar()</i>         | – return the next character from the standard input stream (ANSI) .....              | 2-239 |
| <i>getcwd()</i>          | – get the current default path (POSIX) .....                                         | 2-240 |
| <i>getenv()</i>          | – get an environment variable (ANSI) .....                                           | 2-240 |
| <i>gethostname()</i>     | – get the symbolic name of this machine .....                                        | 2-241 |
| <i>getpeername()</i>     | – get the name of a connected peer .....                                             | 2-241 |
| <i>gets()</i>            | – read characters from the standard input stream (ANSI) .....                        | 2-242 |
| <i>getsockname()</i>     | – get a socket name .....                                                            | 2-242 |
| <i>getsockopt()</i>      | – get socket options .....                                                           | 2-243 |
| <i>getw()</i>            | – read the next word (32-bit integer) from a stream .....                            | 2-244 |
| <i>getwd()</i>           | – get the current default path .....                                                 | 2-244 |
| <i>gmtime()</i>          | – convert calendar time into UTC broken-down time (ANSI) .....                       | 2-245 |
| <i>gmtime_r()</i>        | – convert calendar time into broken-down time (POSIX) .....                          | 2-245 |
| <i>h()</i>               | – display or set the size of shell history .....                                     | 2-246 |
| <i>hdrBlkBuild()</i>     | – create the header block and the demuxer information .....                          | 2-246 |
| <i>help()</i>            | – print a synopsis of selected routines .....                                        | 2-247 |
| <i>hostAdd()</i>         | – add a host to the host table .....                                                 | 2-249 |
| <i>hostDelete()</i>      | – delete a host from the host table .....                                            | 2-250 |
| <i>hostGetByAddr()</i>   | – look up a host in the host table by its Internet address .....                     | 2-250 |
| <i>hostGetByName()</i>   | – look up a host in the host table by its name .....                                 | 2-251 |
| <i>hostShow()</i>        | – display the host table .....                                                       | 2-251 |

|                            |                                                                           |       |
|----------------------------|---------------------------------------------------------------------------|-------|
| <i>hostTblInit()</i>       | – initialize the network host table .....                                 | 2-251 |
| <i>i()</i>                 | – print a summary of each task's TCB .....                                | 2-252 |
| <i>i0()</i>                | – return the contents of register i0 (also i1 – i7) (SPARC) .....         | 2-253 |
| <i>i8250HrdInit()</i>      | – initialize the chip .....                                               | 2-253 |
| <i>i8250Int()</i>          | – handle a receiver/transmitter interrupt .....                           | 2-254 |
| <i>iam()</i>               | – set the remote user name and password .....                             | 2-254 |
| <i>icmpShowInit()</i>      | – initialize ICMP show routines .....                                     | 2-255 |
| <i>icmpstatShow()</i>      | – display statistics for ICMP .....                                       | 2-255 |
| <i>ideDevCreate()</i>      | – create a device for a IDE disk .....                                    | 2-256 |
| <i>ideDrv()</i>            | – initialize the IDE driver .....                                         | 2-256 |
| <i>ideRawio()</i>          | – provide raw I/O access .....                                            | 2-257 |
| <i>ifAddrAdd()</i>         | – Add an interface address for a network interface .....                  | 2-258 |
| <i>ifAddrGet()</i>         | – get the Internet address of a network interface .....                   | 2-258 |
| <i>ifAddrSet()</i>         | – set an interface address for a network interface .....                  | 2-259 |
| <i>ifBroadcastGet()</i>    | – get the broadcast address for a network interface .....                 | 2-259 |
| <i>ifBroadcastSet()</i>    | – set the broadcast address for a network interface .....                 | 2-260 |
| <i>ifDstAddrGet()</i>      | – get the Internet address of a point-to-point peer .....                 | 2-260 |
| <i>ifDstAddrSet()</i>      | – define an address for the other end of a point-to-point link .....      | 2-261 |
| <i>ifFlagChange()</i>      | – change the network interface flags .....                                | 2-261 |
| <i>ifFlagGet()</i>         | – get the network interface flags .....                                   | 2-262 |
| <i>ifFlagSet()</i>         | – specify the flags for a network interface .....                         | 2-262 |
| <i>ifMaskGet()</i>         | – get the subnet mask for a network interface .....                       | 2-263 |
| <i>ifMaskSet()</i>         | – define a subnet for a network interface .....                           | 2-264 |
| <i>ifMetricGet()</i>       | – get the metric for a network interface .....                            | 2-264 |
| <i>ifMetricSet()</i>       | – specify a network interface hop count .....                             | 2-265 |
| <i>ifRouteDelete()</i>     | – delete routes associated with a network interface .....                 | 2-265 |
| <i>ifShow()</i>            | – display the attached network interfaces .....                           | 2-266 |
| <i>ifunit()</i>            | – map an interface name to an interface structure pointer .....           | 2-266 |
| <i>igmpShowInit()</i>      | – initialize IGMP show routines .....                                     | 2-267 |
| <i>igmpstatShow()</i>      | – display statistics for IGMP .....                                       | 2-267 |
| <i>index()</i>             | – find the first occurrence of a character in a string .....              | 2-268 |
| <i>inet_addr()</i>         | – convert a dot notation Internet address to a long integer .....         | 2-268 |
| <i>inet_aton()</i>         | – convert a network address from dot notation, store in a structure ..... | 2-269 |
| <i>inet_lnaof()</i>        | – get the local address (host number) from the Internet address .....     | 2-269 |
| <i>inet_makeaddr()</i>     | – form an Internet address from network and host numbers .....            | 2-270 |
| <i>inet_makeaddr_b()</i>   | – form an Internet address from network and host numbers .....            | 2-270 |
| <i>inet_netof()</i>        | – return the network number from an Internet address .....                | 2-271 |
| <i>inet_netof_string()</i> | – extract the network address in dot notation .....                       | 2-271 |
| <i>inet_network()</i>      | – convert an Internet network number from string to address .....         | 2-272 |
| <i>inet_ntoa()</i>         | – convert a network address to dotted decimal notation .....              | 2-272 |
| <i>inet_ntoa_b()</i>       | – convert an network address to dot notation, store it in a buffer .....  | 2-273 |
| <i>inetstatShow()</i>      | – display all active connections for Internet protocol sockets .....      | 2-274 |
| <i>infinity()</i>          | – return a very large double .....                                        | 2-274 |
| <i>infinityf()</i>         | – return a very large float .....                                         | 2-274 |
| <i>inflate()</i>           | – inflate compressed code .....                                           | 2-275 |
| <i>intConnect()</i>        | – connect a C routine to a hardware interrupt .....                       | 2-275 |
| <i>intContext()</i>        | – determine if the current state is in interrupt or task context .....    | 2-276 |
| <i>intCount()</i>          | – get the current interrupt nesting depth .....                           | 2-277 |



|                                  |                                                                                 |       |
|----------------------------------|---------------------------------------------------------------------------------|-------|
| <i>intCRGet()</i>                | – read the contents of the cause register (MIPS) .....                          | 2-277 |
| <i>intCRSet()</i>                | – write the contents of the cause register (MIPS) .....                         | 2-277 |
| <i>intDisable()</i>              | – disable corresponding interrupt bits (MIPS, PowerPC, ARM) .....               | 2-278 |
| <i>intEnable()</i>               | – enable corresponding interrupt bits (MIPS, PowerPC, ARM) .....                | 2-278 |
| <i>intHandlerCreate()</i>        | – construct interrupt handler for C routine (MC680x0, SPARC, i960, x86, MIPS)   | 2-279 |
| <i>intLevelSet()</i>             | – set the interrupt level (MC680x0, SPARC, i960, x86, ARM) .....                | 2-279 |
| <i>intLock()</i>                 | – lock out interrupts .....                                                     | 2-280 |
| <i>intLockLevelGet()</i>         | – get the current interrupt lock-out level (MC680x0, SPARC, i960, x86, ARM) ... | 2-282 |
| <i>intLockLevelSet()</i>         | – set the current interrupt lock-out level (MC680x0, SPARC, i960, x86, ARM) ... | 2-282 |
| <i>intSRGet()</i>                | – read the contents of the status register (MIPS) .....                         | 2-283 |
| <i>intSRSet()</i>                | – update the contents of the status register (MIPS) .....                       | 2-283 |
| <i>intUninitVecSet()</i>         | – set the uninitialized vector handler (ARM) .....                              | 2-283 |
| <i>intUnlock()</i>               | – cancel interrupt locks .....                                                  | 2-284 |
| <i>intVecBaseGet()</i>           | – get the vector (trap) base address (MC680x0, SPARC, i960, x86, MIPS, ARM) .   | 2-284 |
| <i>intVecBaseSet()</i>           | – set the vector (trap) base address (MC680x0, SPARC, i960, x86, MIPS, ARM) ..  | 2-285 |
| <i>intVecGet()</i>               | – get an interrupt vector (MC680x0, SPARC, i960, x86, MIPS) .....               | 2-286 |
| <i>intVecSet()</i>               | – set a CPU vector (trap) (MC680x0, SPARC, i960, x86, MIPS) .....               | 2-286 |
| <i>intVecTableWriteProtect()</i> | – write-protect exception vector table (MC680x0, SPARC, i960, x86, ARM) .....   | 2-288 |
| <i>ioctl()</i>                   | – perform an I/O control function .....                                         | 2-288 |
| <i>ioDefPathGet()</i>            | – get the current default path .....                                            | 2-289 |
| <i>ioDefPathSet()</i>            | – set the current default path .....                                            | 2-289 |
| <i>ioGlobalStdGet()</i>          | – get the file descriptor for global standard input/output/error .....          | 2-290 |
| <i>ioGlobalStdSet()</i>          | – set the file descriptor for global standard input/output/error .....          | 2-290 |
| <i>ioOlicomEndLoad()</i>         | – initialize the driver and device .....                                        | 2-291 |
| <i>ioOlicomIntHandle()</i>       | – interrupt service for card interrupts .....                                   | 2-291 |
| <i>ioMmuMicroSparcInit()</i>     | – initialize the microSparc I/II I/O MMU data structures .....                  | 2-292 |
| <i>ioMmuMicroSparcMap()</i>      | – map the I/O MMU for microSparc I/II (TMS390S10/MB86904) .....                 | 2-292 |
| <i>iosDevAdd()</i>               | – add a device to the I/O system .....                                          | 2-293 |
| <i>iosDevDelete()</i>            | – delete a device from the I/O system .....                                     | 2-293 |
| <i>iosDevFind()</i>              | – find an I/O device in the device list .....                                   | 2-294 |
| <i>iosDevShow()</i>              | – display the list of devices in the system .....                               | 2-294 |
| <i>iosDrvInstall()</i>           | – install an I/O driver .....                                                   | 2-295 |
| <i>iosDrvRemove()</i>            | – remove an I/O driver .....                                                    | 2-295 |
| <i>iosDrvShow()</i>              | – display a list of system drivers .....                                        | 2-296 |
| <i>iosFdShow()</i>               | – display a list of file descriptor names in the system .....                   | 2-296 |
| <i>iosFdValue()</i>              | – validate an open file descriptor and return the driver-specific value .....   | 2-296 |
| <i>iosInit()</i>                 | – initialize the I/O system .....                                               | 2-297 |
| <i>iosShowInit()</i>             | – initialize the I/O system show facility .....                                 | 2-297 |
| <i>ioTaskStdGet()</i>            | – get the file descriptor for task standard input/output/error .....            | 2-298 |
| <i>ioTaskStdSet()</i>            | – set the file descriptor for task standard input/output/error .....            | 2-298 |
| <i>ipAttach()</i>                | – a generic attach routine for the TCP/IP network stack .....                   | 2-299 |
| <i>ipDetach()</i>                | – a generic detach routine for the TCP/IP network stack .....                   | 2-299 |
| <i>ipFilterHookAdd()</i>         | – add a routine to receive all internet protocol packets .....                  | 2-300 |
| <i>ipFilterHookDelete()</i>      | – delete a ip filter hook routine .....                                         | 2-301 |
| <i>ipFilterLibInit()</i>         | – initialize ip filter facility .....                                           | 2-301 |
| <i>ipstatShow()</i>              | – display IP statistics .....                                                   | 2-301 |
| <i>irint()</i>                   | – convert a double-precision value to an integer .....                          | 2-302 |
| <i>irintf()</i>                  | – convert a single-precision value to an integer .....                          | 2-302 |

|                          |                                                                                     |       |
|--------------------------|-------------------------------------------------------------------------------------|-------|
| <i>iround()</i>          | – round a number to the nearest integer .....                                       | 2-303 |
| <i>iroundf()</i>         | – round a number to the nearest integer .....                                       | 2-303 |
| <i>isalnum()</i>         | – test whether a character is alphanumeric (ANSI) .....                             | 2-304 |
| <i>isalpha()</i>         | – test whether a character is a letter (ANSI) .....                                 | 2-304 |
| <i>isatty()</i>          | – return whether the underlying driver is a tty device .....                        | 2-305 |
| <i>iscntrl()</i>         | – test whether a character is a control character (ANSI) .....                      | 2-305 |
| <i>isdigit()</i>         | – test whether a character is a decimal digit (ANSI) .....                          | 2-306 |
| <i>isgraph()</i>         | – test whether a character is a printing, non-white-space character (ANSI) .....    | 2-306 |
| <i>islower()</i>         | – test whether a character is a lower-case letter (ANSI) .....                      | 2-307 |
| <i>isprint()</i>         | – test whether a character is printable, including the space character (ANSI) ..... | 2-307 |
| <i>ispunct()</i>         | – test whether a character is punctuation (ANSI) .....                              | 2-308 |
| <i>isspace()</i>         | – test whether a character is a white-space character (ANSI) .....                  | 2-308 |
| <i>isupper()</i>         | – test whether a character is an upper-case letter (ANSI) .....                     | 2-309 |
| <i>isxdigit()</i>        | – test whether a character is a hexadecimal digit (ANSI) .....                      | 2-309 |
| <i>kernelInit()</i>      | – initialize the kernel .....                                                       | 2-310 |
| <i>kernelTimeSlice()</i> | – enable round-robin selection .....                                                | 2-310 |
| <i>kernelVersion()</i>   | – return the kernel revision string .....                                           | 2-311 |
| <i>kill()</i>            | – send a signal to a task (POSIX) .....                                             | 2-311 |
| <i>ll()</i>              | – disassemble and display a specified number of instructions .....                  | 2-312 |
| <i>l0()</i>              | – return the contents of register l0 (also l1 – l7) (SPARC) .....                   | 2-312 |
| <i>labs()</i>            | – compute the absolute value of a long (ANSI) .....                                 | 2-313 |
| <i>ld()</i>              | – load an object module into memory .....                                           | 2-313 |
| <i>ldexp()</i>           | – multiply a number by an integral power of 2 (ANSI) .....                          | 2-314 |
| <i>ldiv()</i>            | – compute the quotient and remainder of the division (ANSI) .....                   | 2-315 |
| <i>ldiv_r()</i>          | – compute a quotient and remainder (reentrant) .....                                | 2-315 |
| <i>ledClose()</i>        | – discard the line-editor ID .....                                                  | 2-316 |
| <i>ledControl()</i>      | – change the line-editor ID parameters .....                                        | 2-316 |
| <i>ledOpen()</i>         | – create a new line-editor ID .....                                                 | 2-317 |
| <i>ledRead()</i>         | – read a line with line-editing .....                                               | 2-317 |
| <i>lio_listio()</i>      | – initiate a list of asynchronous I/O requests (POSIX) .....                        | 2-318 |
| <i>listen()</i>          | – enable connections to a socket .....                                              | 2-319 |
| <i>lkAddr()</i>          | – list symbols whose values are near a specified value .....                        | 2-319 |
| <i>lkup()</i>            | – list symbols .....                                                                | 2-320 |
| <i>ll()</i>              | – do a long listing of directory contents .....                                     | 2-320 |
| <i>ln97xEndLoad()</i>    | – initialize the driver and device .....                                            | 2-321 |
| <i>ln97xInitParse()</i>  | – parse the initialization string .....                                             | 2-321 |
| <i>ln7990EndLoad()</i>   | – initialize the driver and device .....                                            | 2-323 |
| <i>ln7990InitMem()</i>   | – initialize memory for Lance chip .....                                            | 2-323 |
| <i>ln7990InitParse()</i> | – parse the initialization string .....                                             | 2-324 |
| <i>lnattach()</i>        | – publish the ln network interface and initialize driver structures .....           | 2-325 |
| <i>lnPciattach()</i>     | – publish the lnPci network interface and initialize the driver and device .....    | 2-326 |
| <i>loadModule()</i>      | – load an object module into memory .....                                           | 2-327 |
| <i>loadModuleAt()</i>    | – load an object module into memory .....                                           | 2-327 |
| <i>loattach()</i>        | – publish the lo network interface and initialize the driver and pseudo-device ..   | 2-330 |
| <i>localeconv()</i>      | – set the components of an object with type lconv (ANSI) .....                      | 2-330 |
| <i>localtime()</i>       | – convert calendar time into broken-down time (ANSI) .....                          | 2-333 |
| <i>localtime_r()</i>     | – convert calendar time into broken-down time (POSIX) .....                         | 2-333 |
| <i>log()</i>             | – compute a natural logarithm (ANSI) .....                                          | 2-334 |

|                              |                                                                      |       |
|------------------------------|----------------------------------------------------------------------|-------|
| <i>log2()</i>                | – compute a base-2 logarithm .....                                   | 2-334 |
| <i>log2f()</i>               | – compute a base-2 logarithm .....                                   | 2-335 |
| <i>log10()</i>               | – compute a base-10 logarithm (ANSI) .....                           | 2-335 |
| <i>log10f()</i>              | – compute a base-10 logarithm (ANSI) .....                           | 2-336 |
| <i>logf()</i>                | – compute a natural logarithm (ANSI) .....                           | 2-336 |
| <i>logFdAdd()</i>            | – add a logging file descriptor .....                                | 2-337 |
| <i>logFdDelete()</i>         | – delete a logging file descriptor .....                             | 2-337 |
| <i>logFdSet()</i>            | – set the primary logging file descriptor .....                      | 2-338 |
| <i>loginDefaultEncrypt()</i> | – default password encryption routine .....                          | 2-338 |
| <i>loginEncryptInstall()</i> | – install an encryption routine .....                                | 2-339 |
| <i>loginInit()</i>           | – initialize the login table .....                                   | 2-340 |
| <i>logInit()</i>             | – initialize message logging library .....                           | 2-340 |
| <i>loginPrompt()</i>         | – display a login prompt and validate a user entry .....             | 2-341 |
| <i>loginStringSet()</i>      | – change the login string .....                                      | 2-341 |
| <i>loginUserAdd()</i>        | – add a user to the login table .....                                | 2-342 |
| <i>loginUserDelete()</i>     | – delete a user entry from the login table .....                     | 2-343 |
| <i>loginUserShow()</i>       | – display the user login table .....                                 | 2-343 |
| <i>loginUserVerify()</i>     | – verify a user name and password in the login table .....           | 2-344 |
| <i>logMsg()</i>              | – log a formatted error message .....                                | 2-344 |
| <i>logout()</i>              | – log out of the VxWorks system .....                                | 2-345 |
| <i>logTask()</i>             | – message-logging support task .....                                 | 2-345 |
| <i>longjmp()</i>             | – perform non-local goto by restoring saved environment (ANSI) ..... | 2-346 |
| <i>lptDevCreate()</i>        | – create a device for an LPT port .....                              | 2-347 |
| <i>lptDrv()</i>              | – initialize the LPT driver .....                                    | 2-347 |
| <i>lptShow()</i>             | – show LPT statistics .....                                          | 2-348 |
| <i>ls()</i>                  | – list the contents of a directory .....                             | 2-348 |
| <i>lseek()</i>               | – set a file read/write pointer .....                                | 2-349 |
| <i>lsOld()</i>               | – list the contents of an RT-11 directory .....                      | 2-349 |
| <i>lstAdd()</i>              | – add a node to the end of a list .....                              | 2-350 |
| <i>lstConcat()</i>           | – concatenate two lists .....                                        | 2-350 |
| <i>lstCount()</i>            | – report the number of nodes in a list .....                         | 2-351 |
| <i>lstDelete()</i>           | – delete a specified node from a list .....                          | 2-351 |
| <i>lstExtract()</i>          | – extract a sublist from a list .....                                | 2-352 |
| <i>lstFind()</i>             | – find a node in a list .....                                        | 2-352 |
| <i>lstFirst()</i>            | – find first node in list .....                                      | 2-353 |
| <i>lstFree()</i>             | – free up a list .....                                               | 2-353 |
| <i>lstGet()</i>              | – delete and return the first node from a list .....                 | 2-354 |
| <i>lstInit()</i>             | – initialize a list descriptor .....                                 | 2-354 |
| <i>lstInsert()</i>           | – insert a node in a list after a specified node .....               | 2-355 |
| <i>lstLast()</i>             | – find the last node in a list .....                                 | 2-355 |
| <i>lstNext()</i>             | – find the next node in a list .....                                 | 2-356 |
| <i>lstNStep()</i>            | – find a list node nStep steps away from a specified node .....      | 2-356 |
| <i>lstNth()</i>              | – find the Nth node in a list .....                                  | 2-357 |
| <i>lstPrevious()</i>         | – find the previous node in a list .....                             | 2-357 |
| <i>m()</i>                   | – modify memory .....                                                | 2-358 |
| <i>m2Delete()</i>            | – delete all the MIB-II library groups .....                         | 2-358 |
| <i>m2IcmpDelete()</i>        | – delete all resources used to access the ICMP group .....           | 2-359 |
| <i>m2IcmpGroupInfoGet()</i>  | – get the MIB-II ICMP-group global variables .....                   | 2-359 |

|                                  |                                                                 |       |
|----------------------------------|-----------------------------------------------------------------|-------|
| <b>m2IcmpInit()</b>              | – initialize MIB-II ICMP-group access .....                     | 2-360 |
| <b>m2IfDelete()</b>              | – delete all resources used to access the interface group ..... | 2-360 |
| <b>m2IfGroupInfoGet()</b>        | – get the MIB-II interface-group scalar variables .....         | 2-360 |
| <b>m2IfInit()</b>                | – initialize MIB-II interface-group routines .....              | 2-361 |
| <b>m2IfTblEntryGet()</b>         | – get a MIB-II interface-group table entry .....                | 2-361 |
| <b>m2IfTblEntrySet()</b>         | – set the state of a MIB-II interface entry to UP or DOWN ..... | 2-362 |
| <b>m2Init()</b>                  | – initialize the SNMP MIB-2 library .....                       | 2-363 |
| <b>m2IpAddrTblEntryGet()</b>     | – get an IP MIB-II address entry .....                          | 2-363 |
| <b>m2IpAtransTblEntryGet()</b>   | – get a MIB-II ARP table entry .....                            | 2-364 |
| <b>m2IpAtransTblEntrySet()</b>   | – add, modify, or delete a MIB-II ARP entry .....               | 2-365 |
| <b>m2IpDelete()</b>              | – delete all resources used to access the IP group .....        | 2-365 |
| <b>m2IpGroupInfoGet()</b>        | – get the MIB-II IP-group scalar variables .....                | 2-366 |
| <b>m2IpGroupInfoSet()</b>        | – set MIB-II IP-group variables to new values .....             | 2-366 |
| <b>m2IpInit()</b>                | – initialize MIB-II IP-group access .....                       | 2-367 |
| <b>m2IpRouteTblEntryGet()</b>    | – get a MIB-2 routing table entry .....                         | 2-367 |
| <b>m2IpRouteTblEntrySet()</b>    | – set a MIB-II routing table entry .....                        | 2-368 |
| <b>m2OspfAreaEntryGet()</b>      | – get an entry from the OSPF area table (OSPF Opt.) .....       | 2-369 |
| <b>m2OspfAreaEntrySet()</b>      | – set values in an OSPF area entry (OSPF Opt.) .....            | 2-369 |
| <b>m2OspfAreaRangeEntryGet()</b> | – get an OSPF area range entry (OSPF Opt.) .....                | 2-370 |
| <b>m2OspfAreaRangeEntrySet()</b> | – set values in an OSPF area range entry (OSPF Opt.) .....      | 2-370 |
| <b>m2OspfGeneralGroupGet()</b>   | – get values of OSPF general group objects (OSPF Opt.) .....    | 2-371 |
| <b>m2OspfGeneralGroupSet()</b>   | – set values of OSPF general group objects (OSPF Opt.) .....    | 2-371 |
| <b>m2OspfHostEntryGet()</b>      | – get an OSPF host entry (OSPF Opt.) .....                      | 2-372 |
| <b>m2OspfHostEntrySet()</b>      | – set values in an OSPF host entry (OSPF Opt.) .....            | 2-372 |
| <b>m2OspfIfEntryGet()</b>        | – get an OSPF interface entry (OSPF Opt.) .....                 | 2-373 |
| <b>m2OspfIfEntrySet()</b>        | – set values in an OSPF interface entry (OSPF Opt.) .....       | 2-373 |
| <b>m2OspfIfMetricEntryGet()</b>  | – get an OSPF interface metric entry (OSPF Opt.) .....          | 2-374 |
| <b>m2OspfIfMetricEntrySet()</b>  | – set OSPF interface metric entry values (OSPF Opt.) .....      | 2-374 |
| <b>m2OspfLsdbEntryGet()</b>      | – get an OSPF link state database entry (OSPF Opt.) .....       | 2-375 |
| <b>m2OspfNbrEntryGet()</b>       | – get an OSPF neighbor entry (OSPF Opt.) .....                  | 2-375 |
| <b>m2OspfNbrEntrySet()</b>       | – set values in an OSPF neighbor entry (OSPF Opt.) .....        | 2-376 |
| <b>m2OspfStubAreaEntryGet()</b>  | – get an OSPF stub area entry (OSPF Opt.) .....                 | 2-376 |
| <b>m2OspfStubAreaEntrySet()</b>  | – set values in an OSPF stub area entry (OSPF Opt.) .....       | 2-377 |
| <b>m2OspfVirtIfEntryGet()</b>    | – get an OSPF virtual interface entry (OSPF Opt.) .....         | 2-377 |
| <b>m2OspfVirtIfEntrySet()</b>    | – set OSPF virtual interface entry values (OSPF Opt.) .....     | 2-378 |
| <b>m2OspfVirtNbrEntryGet()</b>   | – get an OSPF virtual neighbor entry (OSPF Opt.) .....          | 2-378 |
| <b>m2SysDelete()</b>             | – delete resources used to access the MIB-II system group ..... | 2-379 |
| <b>m2SysGroupInfoGet()</b>       | – get system-group MIB-II variables .....                       | 2-379 |
| <b>m2SysGroupInfoSet()</b>       | – set system-group MIB-II variables to new values .....         | 2-380 |
| <b>m2SysInit()</b>               | – initialize MIB-II system-group routines .....                 | 2-380 |
| <b>m2TcpConnEntryGet()</b>       | – get a MIB-II TCP connection table entry .....                 | 2-381 |
| <b>m2TcpConnEntrySet()</b>       | – set a TCP connection to the closed state .....                | 2-381 |
| <b>m2TcpDelete()</b>             | – delete all resources used to access the TCP group .....       | 2-382 |
| <b>m2TcpGroupInfoGet()</b>       | – get MIB-II TCP-group scalar variables .....                   | 2-382 |
| <b>m2TcpInit()</b>               | – initialize MIB-II TCP-group access .....                      | 2-383 |
| <b>m2UdpDelete()</b>             | – delete all resources used to access the UDP group .....       | 2-383 |
| <b>m2UdpGroupInfoGet()</b>       | – get MIB-II UDP-group scalar variables .....                   | 2-384 |
| <b>m2UdpInit()</b>               | – initialize MIB-II UDP-group access .....                      | 2-384 |

|                            |                                                                            |       |
|----------------------------|----------------------------------------------------------------------------|-------|
| <i>m2UdpTblEntryGet()</i>  | - get a UDP MIB-II entry from the UDP list of listeners .....              | 2-384 |
| <i>m68302SioInit()</i>     | - initialize a M68302_CP .....                                             | 2-385 |
| <i>m68302SioInit2()</i>    | - initialize a M68302_CP (part 2) .....                                    | 2-386 |
| <i>m68332DevInit()</i>     | - initialize the SCC .....                                                 | 2-386 |
| <i>m68332Int()</i>         | - handle an SCC interrupt .....                                            | 2-387 |
| <i>m68360DevInit()</i>     | - initialize the SCC .....                                                 | 2-387 |
| <i>m68360Int()</i>         | - handle an SCC interrupt .....                                            | 2-387 |
| <i>m68562HrdInit()</i>     | - initialize the DUSCC .....                                               | 2-388 |
| <i>m68562RxInt()</i>       | - handle a receiver interrupt .....                                        | 2-388 |
| <i>m68562RxTxErrInt()</i>  | - handle a receiver/transmitter error interrupt .....                      | 2-389 |
| <i>m68562TxInt()</i>       | - handle a transmitter interrupt .....                                     | 2-389 |
| <i>m68681Acr()</i>         | - return the contents of the DUART auxiliary control register .....        | 2-390 |
| <i>m68681AcrSetClr()</i>   | - set and clear bits in the DUART auxiliary control register .....         | 2-390 |
| <i>m68681DevInit()</i>     | - initialize a M68681_DUART .....                                          | 2-391 |
| <i>m68681DevInit2()</i>    | - initialize a M68681_DUART, part 2 .....                                  | 2-391 |
| <i>m68681Imr()</i>         | - return the current contents of the DUART interrupt-mask register .....   | 2-392 |
| <i>m68681ImrSetClr()</i>   | - set and clear bits in the DUART interrupt-mask register .....            | 2-392 |
| <i>m68681Int()</i>         | - handle all DUART interrupts in one vector .....                          | 2-393 |
| <i>m68681Opcr()</i>        | - return the state of the DUART output port configuration register .....   | 2-393 |
| <i>m68681OpcrSetClr()</i>  | - set and clear bits in the DUART output port configuration register ..... | 2-394 |
| <i>m68681Opr()</i>         | - return the current state of the DUART output port register .....         | 2-394 |
| <i>m68681OprSetClr()</i>   | - set and clear bits in the DUART output port register .....               | 2-395 |
| <i>m68901DevInit()</i>     | - initialize a M68901_CHAN structure .....                                 | 2-395 |
| <i>malloc()</i>            | - allocate a block of memory from the system memory partition (ANSI) ..... | 2-396 |
| <i>masterIoInit()</i>      | - create the IPC mechanism at the SNMP master agent .....                  | 2-396 |
| <i>masterIoWrite()</i>     | - send the encoded buffer to the subagent .....                            | 2-397 |
| <i>masterIpcAyt()</i>      | - check the status of the IPC link .....                                   | 2-398 |
| <i>masterIpcComp()</i>     | - transmit a completion of transmission message .....                      | 2-398 |
| <i>masterIpcFree()</i>     | - free the IPC resources allocated by the SNMP master agent .....          | 2-399 |
| <i>masterIpcRcv()</i>      | - wait for a reply from the subagent .....                                 | 2-400 |
| <i>masterIpcSend()</i>     | - send a message to a subagent .....                                       | 2-401 |
| <i>masterQueCleanup()</i>  | - free resources allocated for SNMP master agent .....                     | 2-401 |
| <i>mathHardInit()</i>      | - initialize hardware floating-point math support .....                    | 2-402 |
| <i>mathSoftInit()</i>      | - initialize software floating-point math support .....                    | 2-402 |
| <i>mb86940DevInit()</i>    | - install the driver function table .....                                  | 2-403 |
| <i>mb86960EndLoad()</i>    | - initialize the driver and device .....                                   | 2-403 |
| <i>mb86960InitParse()</i>  | - parse the initialization string .....                                    | 2-404 |
| <i>mb86960MemInit()</i>    | - initialize memory for the chip .....                                     | 2-405 |
| <i>mb87030CtrlCreate()</i> | - create a control structure for an MB87030 SPC .....                      | 2-405 |
| <i>mb87030CtrlInit()</i>   | - initialize a control structure for an MB87030 SPC .....                  | 2-406 |
| <i>mb87030Show()</i>       | - display the values of all readable MB87030 SPC registers .....           | 2-407 |
| <i>mbcAddrFilterSet()</i>  | - set the address filter for multicast addresses .....                     | 2-408 |
| <i>mbcattach()</i>         | - publish the mbc network interface and initialize the driver .....        | 2-408 |
| <i>mbcEndLoad()</i>        | - initialize the driver and device .....                                   | 2-409 |
| <i>mbcIntr()</i>           | - network interface interrupt handler .....                                | 2-410 |
| <i>mbcMemInit()</i>        | - initialize memory for the chip .....                                     | 2-410 |
| <i>mbcParse()</i>          | - parse the init string .....                                              | 2-411 |
| <i>mbcStartOutput()</i>    | - output packet to network interface device .....                          | 2-412 |

|                                         |                                                                               |       |
|-----------------------------------------|-------------------------------------------------------------------------------|-------|
| <b><i>mblen()</i></b>                   | – calculate the length of a multibyte character (Unimplemented) (ANSI) .....  | 2-413 |
| <b><i>mbstowcs()</i></b>                | – convert a series of multibyte char's to wide char's (Unimplemented) (ANSI)  | 2-413 |
| <b><i>mbtowc()</i></b>                  | – convert a multibyte character to a wide character (Unimplemented) (ANSI)    | 2-414 |
| <b><i>mbufShow()</i></b>                | – report mbuf statistics .....                                                | 2-414 |
| <b><i>memAddToPool()</i></b>            | – add memory to the system memory partition .....                             | 2-415 |
| <b><i>memalign()</i></b>                | – allocate aligned memory .....                                               | 2-415 |
| <b><i>memchr()</i></b>                  | – search a block of memory for a character (ANSI) .....                       | 2-416 |
| <b><i>memcmp()</i></b>                  | – compare two blocks of memory (ANSI) .....                                   | 2-416 |
| <b><i>memcpy()</i></b>                  | – copy memory from one location to another (ANSI) .....                       | 2-417 |
| <b><i>memDevCreate()</i></b>            | – create a memory device .....                                                | 2-417 |
| <b><i>memDevCreateDir()</i></b>         | – create a memory device for multiple files .....                             | 2-419 |
| <b><i>memDevDelete()</i></b>            | – delete a memory device .....                                                | 2-419 |
| <b><i>memDrv()</i></b>                  | – install a memory driver .....                                               | 2-420 |
| <b><i>memFindMax()</i></b>              | – find the largest free block in the system memory partition .....            | 2-420 |
| <b><i>memmove()</i></b>                 | – copy memory from one location to another (ANSI) .....                       | 2-421 |
| <b><i>memOptionsSet()</i></b>           | – set the debug options for the system memory partition .....                 | 2-421 |
| <b><i>memPartAddToPool()</i></b>        | – add memory to a memory partition .....                                      | 2-422 |
| <b><i>memPartAlignedAlloc()</i></b>     | – allocate aligned memory from a partition .....                              | 2-422 |
| <b><i>memPartAlloc()</i></b>            | – allocate a block of memory from a partition .....                           | 2-423 |
| <b><i>memPartCreate()</i></b>           | – create a memory partition .....                                             | 2-423 |
| <b><i>memPartFindMax()</i></b>          | – find the size of the largest available free block .....                     | 2-424 |
| <b><i>memPartFree()</i></b>             | – free a block of memory in a partition .....                                 | 2-424 |
| <b><i>memPartInfoGet()</i></b>          | – get partition information .....                                             | 2-425 |
| <b><i>memPartOptionsSet()</i></b>       | – set the debug options for a memory partition .....                          | 2-425 |
| <b><i>memPartRealloc()</i></b>          | – reallocate a block of memory in a specified partition .....                 | 2-426 |
| <b><i>memPartShow()</i></b>             | – show partition blocks and statistics .....                                  | 2-427 |
| <b><i>memPartSmCreate()</i></b>         | – create a shared memory partition (VxMP Opt.) .....                          | 2-427 |
| <b><i>memset()</i></b>                  | – set a block of memory (ANSI) .....                                          | 2-428 |
| <b><i>memShow()</i></b>                 | – show system memory partition blocks and statistics .....                    | 2-429 |
| <b><i>memShowInit()</i></b>             | – initialize the memory partition show facility .....                         | 2-430 |
| <b><i>mib2ErrorAdd()</i></b>            | – change a MIB-II error count .....                                           | 2-430 |
| <b><i>mib2Init()</i></b>                | – initialize a MIB-II structure .....                                         | 2-431 |
| <b><i>mkdir()</i></b>                   | – make a directory .....                                                      | 2-431 |
| <b><i>mktime()</i></b>                  | – convert broken-down time into calendar time (ANSI) .....                    | 2-432 |
| <b><i>mlock()</i></b>                   | – lock specified pages into memory (POSIX) .....                              | 2-432 |
| <b><i>mlockall()</i></b>                | – lock all pages used by a process into memory (POSIX) .....                  | 2-433 |
| <b><i>mmuL64862DmaInit()</i></b>        | – initialize the L64862 I/O MMU DMA data structures (SPARC) .....             | 2-433 |
| <b><i>mmuPro32LibInit()</i></b>         | – initialize module .....                                                     | 2-434 |
| <b><i>mmuSparcRomInit()</i></b>         | – initialize the MMU for the ROM (SPARC) .....                                | 2-434 |
| <b><i>modf()</i></b>                    | – separate a floating-point number into integer and fraction parts (ANSI) ... | 2-435 |
| <b><i>moduleCheck()</i></b>             | – verify checksums on all modules .....                                       | 2-435 |
| <b><i>moduleCreate()</i></b>            | – create and initialize a module .....                                        | 2-436 |
| <b><i>moduleCreateHookAdd()</i></b>     | – add a routine to be called when a module is added .....                     | 2-437 |
| <b><i>moduleCreateHookDelete()</i></b>  | – delete a previously added module create hook routine .....                  | 2-437 |
| <b><i>moduleDelete()</i></b>            | – delete module ID information (use <i>unld()</i> to reclaim space) .....     | 2-438 |
| <b><i>moduleFindByGroup()</i></b>       | – find a module by group number .....                                         | 2-438 |
| <b><i>moduleFindByName()</i></b>        | – find a module by name .....                                                 | 2-439 |
| <b><i>moduleFindByNameAndPath()</i></b> | – find a module by file name and path .....                                   | 2-439 |

|                            |                                                                         |       |
|----------------------------|-------------------------------------------------------------------------|-------|
| <i>moduleFlagsGet()</i>    | – get the flags associated with a module ID .....                       | 2-440 |
| <i>moduleIdListGet()</i>   | – get a list of loaded modules .....                                    | 2-440 |
| <i>moduleInfoGet()</i>     | – get information about an object module .....                          | 2-441 |
| <i>moduleNameGet()</i>     | – get the name associated with a module ID .....                        | 2-441 |
| <i>moduleSegFirst()</i>    | – find the first segment in a module .....                              | 2-442 |
| <i>moduleSegGet()</i>      | – get (delete and return) the first segment from a module .....         | 2-442 |
| <i>moduleSegNext()</i>     | – find the next segment in a module .....                               | 2-443 |
| <i>moduleShow()</i>        | – show the current status for all the loaded modules .....              | 2-443 |
| <i>motCpmEndLoad()</i>     | – initialize the driver and device .....                                | 2-444 |
| <i>motFecEndLoad()</i>     | – initialize the driver and device .....                                | 2-445 |
| <i>mountdInit()</i>        | – initialize the mount daemon .....                                     | 2-446 |
| <i>mqPxLibInit()</i>       | – initialize the POSIX message queue library .....                      | 2-447 |
| <i>mqPxShowInit()</i>      | – initialize the POSIX message queue show facility .....                | 2-447 |
| <i>mq_close()</i>          | – close a message queue (POSIX) .....                                   | 2-448 |
| <i>mq_getattr()</i>        | – get message queue attributes (POSIX) .....                            | 2-448 |
| <i>mq_notify()</i>         | – notify a task that a message is available on a queue (POSIX) .....    | 2-449 |
| <i>mq_open()</i>           | – open a message queue (POSIX) .....                                    | 2-450 |
| <i>mq_receive()</i>        | – receive a message from a message queue (POSIX) .....                  | 2-451 |
| <i>mq_send()</i>           | – send a message to a message queue (POSIX) .....                       | 2-452 |
| <i>mq_setattr()</i>        | – set message queue attributes (POSIX) .....                            | 2-453 |
| <i>mq_unlink()</i>         | – remove a message queue (POSIX) .....                                  | 2-454 |
| <i>mRegs()</i>             | – modify registers .....                                                | 2-454 |
| <i>mRouteAdd()</i>         | – add multiple routes to the same destination .....                     | 2-455 |
| <i>mRouteDelete()</i>      | – delete a route from the routing table .....                           | 2-456 |
| <i>mRouteEntryAdd()</i>    | – add a protocol-specific route to the routing table .....              | 2-457 |
| <i>mRouteEntryDelete()</i> | – delete route from the routing table .....                             | 2-458 |
| <i>mRouteShow()</i>        | – print the entries of the routing table .....                          | 2-458 |
| <i>msgQCreate()</i>        | – create and initialize a message queue .....                           | 2-459 |
| <i>msgQDelete()</i>        | – delete a message queue .....                                          | 2-459 |
| <i>msgQInfoGet()</i>       | – get information about a message queue .....                           | 2-460 |
| <i>msgQNumMsgs()</i>       | – get the number of messages queued to a message queue .....            | 2-462 |
| <i>msgQReceive()</i>       | – receive a message from a message queue .....                          | 2-462 |
| <i>msgQSend()</i>          | – send a message to a message queue .....                               | 2-463 |
| <i>msgQShow()</i>          | – show information about a message queue .....                          | 2-464 |
| <i>msgQShowInit()</i>      | – initialize the message queue show facility .....                      | 2-465 |
| <i>msgQSmCreate()</i>      | – create and initialize a shared memory message queue (VxMP Opt.) ..... | 2-466 |
| <i>munlock()</i>           | – unlock specified pages (POSIX) .....                                  | 2-467 |
| <i>munlockall()</i>        | – unlock all pages used by a process (POSIX) .....                      | 2-467 |
| <i>muxAddressForm()</i>    | – form an address into a packet .....                                   | 2-468 |
| <i>muxAddrResFuncAdd()</i> | – add an address resolution function .....                              | 2-469 |
| <i>muxAddrResFuncDel()</i> | – delete an address resolution function .....                           | 2-469 |
| <i>muxAddrResFuncGet()</i> | – get the address resolution function for ifType/protocol .....         | 2-470 |
| <i>muxBind()</i>           | – bind a protocol to the MUX given a driver name .....                  | 2-471 |
| <i>muxDevExists()</i>      | – tests whether a device is already loaded into the MUX .....           | 2-472 |
| <i>muxDevLoad()</i>        | – load a driver into the MUX .....                                      | 2-473 |
| <i>muxDevStart()</i>       | – start a device by calling its start routine .....                     | 2-474 |
| <i>muxDevStop()</i>        | – stop a device by calling its stop routine .....                       | 2-474 |
| <i>muxDevUnload()</i>      | – remove a driver from the MUX .....                                    | 2-475 |

|                                   |                                                                          |       |
|-----------------------------------|--------------------------------------------------------------------------|-------|
| <i>muxIoctl()</i>                 | – send control information to the MUX or to a device .....               | 2-476 |
| <i>muxLibInit()</i>               | – initialize global state for the MUX .....                              | 2-477 |
| <i>muxMCastAddrAdd()</i>          | – add a multicast address to multicast table for a device .....          | 2-477 |
| <i>muxMCastAddrDel()</i>          | – delete a multicast address from a device’s multicast table .....       | 2-478 |
| <i>muxMCastAddrGet()</i>          | – get the multicast address table from the MUX/Driver .....              | 2-478 |
| <i>muxPacketAddrGet()</i>         | – get addressing information from a packet .....                         | 2-479 |
| <i>muxPacketDataGet()</i>         | – return the data from a packet .....                                    | 2-480 |
| <i>muxPollReceive()</i>           | – poll for a packet from a device driver .....                           | 2-481 |
| <i>muxPollSend()</i>              | – send a packet on a network interface .....                             | 2-482 |
| <i>muxSend()</i>                  | – send a packet out on a network interface .....                         | 2-482 |
| <i>muxShow()</i>                  | – all configured Enhanced Network Drivers .....                          | 2-483 |
| <i>muxUnbind()</i>                | – detach a protocol from the specified driver .....                      | 2-484 |
| <i>nanosleep()</i>                | – suspend the current task until the time interval elapses (POSIX) ..... | 2-484 |
| <i>ncr710CtrlCreate()</i>         | – create a control structure for an NCR 53C710 SIOP .....                | 2-485 |
| <i>ncr710CtrlCreateScsi2()</i>    | – create a control structure for the NCR 53C710 SIOP .....               | 2-486 |
| <i>ncr710CtrlInit()</i>           | – initialize a control structure for an NCR 53C710 SIOP .....            | 2-487 |
| <i>ncr710CtrlInitScsi2()</i>      | – initialize a control structure for the NCR 53C710 SIOP .....           | 2-488 |
| <i>ncr710SetHwRegister()</i>      | – set hardware-dependent registers for the NCR 53C710 SIOP .....         | 2-489 |
| <i>ncr710SetHwRegisterScsi2()</i> | – set hardware-dependent registers for the NCR 53C710 .....              | 2-490 |
| <i>ncr710Show()</i>               | – display the values of all readable NCR 53C710 SIOP registers .....     | 2-491 |
| <i>ncr710ShowScsi2()</i>          | – display the values of all readable NCR 53C710 SIOP registers .....     | 2-492 |
| <i>ncr810CtrlCreate()</i>         | – create a control structure for the NCR 53C8xx SIOP .....               | 2-493 |
| <i>ncr810CtrlInit()</i>           | – initialize a control structure for the NCR 53C8xx SIOP .....           | 2-494 |
| <i>ncr810SetHwRegister()</i>      | – set hardware-dependent registers for the NCR 53C8xx SIOP .....         | 2-495 |
| <i>ncr810Show()</i>               | – display values of all readable NCR 53C8xx SIOP registers .....         | 2-496 |
| <i>ncr5390CtrlCreate()</i>        | – create a control structure for an NCR 53C90 ASC .....                  | 2-497 |
| <i>ncr5390CtrlCreateScsi2()</i>   | – create a control structure for an NCR 53C90 ASC .....                  | 2-498 |
| <i>ncr5390CtrlInit()</i>          | – initialize the user-specified fields in an ASC structure .....         | 2-499 |
| <i>ncr5390Show()</i>              | – display the values of all readable NCR5390 chip registers .....        | 2-500 |
| <i>ncr710SingleStep()</i>         | – perform a single-step .....                                            | 2-501 |
| <i>ncr710StepEnable()</i>         | – enable/disable script single-step .....                                | 2-501 |
| <i>ne2000EndLoad()</i>            | – initialize the driver and device .....                                 | 2-502 |
| <i>ne2000Parse()</i>              | – parse the init string .....                                            | 2-502 |
| <i>netBufLibInit()</i>            | – initialize <b>netBufLib</b> .....                                      | 2-503 |
| <i>netCIBlkFree()</i>             | – free a cIBlk-cluster construct back to the memory pool .....           | 2-504 |
| <i>netCIBlkGet()</i>              | – get a cIBlk .....                                                      | 2-504 |
| <i>netCIBlkJoin()</i>             | – join a cluster to a cIBlk structure .....                              | 2-505 |
| <i>netCIFree()</i>                | – free a cluster back to the memory pool .....                           | 2-505 |
| <i>netCIPoolIdGet()</i>           | – return a <b>CL_POOL_ID</b> for a specified buffer size .....           | 2-506 |
| <i>netClusterGet()</i>            | – get a cluster from the specified cluster pool .....                    | 2-506 |
| <i>netDevCreate()</i>             | – create a remote file device .....                                      | 2-507 |
| <i>netDrv()</i>                   | – install the network remote file driver .....                           | 2-507 |
| <i>netHelp()</i>                  | – print a synopsis of network routines .....                             | 2-508 |
| <i>netLibInit()</i>               | – initialize the network package .....                                   | 2-509 |
| <i>netMblkChainDup()</i>          | – duplicate an mBlk chain .....                                          | 2-509 |
| <i>netMblkCIBlkChainFree()</i>    | – free a chain of mBlk-cIBlk-cluster constructs .....                    | 2-510 |
| <i>netMblkCIFree()</i>            | – free an mBlk-cIBlk-cluster construct .....                             | 2-510 |
| <i>netMblkCIGet()</i>             | – get a cIBlk-cluster and join it to the specified mBlk .....            | 2-511 |



|                               |                                                                      |       |
|-------------------------------|----------------------------------------------------------------------|-------|
| <i>netMblkClJoin()</i>        | – join an mBlk to a cBlk-cluster construct .....                     | 2-512 |
| <i>netMblkDup()</i>           | – duplicate an mBlk .....                                            | 2-513 |
| <i>netMblkFree()</i>          | – free an mBlk back to its memory pool .....                         | 2-513 |
| <i>netMblkGet()</i>           | – get an mBlk .....                                                  | 2-514 |
| <i>netMblkToBufCopy()</i>     | – copy data from an mBlk to a buffer .....                           | 2-514 |
| <i>netPoolDelete()</i>        | – delete a memory pool .....                                         | 2-515 |
| <i>netPoolInit()</i>          | – initialize a netBufLib-managed memory pool .....                   | 2-515 |
| <i>netPoolShow()</i>          | – show pool statistics .....                                         | 2-518 |
| <i>netShowInit()</i>          | – initialize network show routines .....                             | 2-519 |
| <i>netStackDataPoolShow()</i> | – show network stack data pool statistics .....                      | 2-519 |
| <i>netStackSysPoolShow()</i>  | – show network stack system pool statistics .....                    | 2-520 |
| <i>netTask()</i>              | – network task entry point .....                                     | 2-520 |
| <i>netTupleGet()</i>          | – get an mBlk-cBlk-cluster .....                                     | 2-521 |
| <i>nfsAuthUnixGet()</i>       | – get the NFS UNIX authentication parameters .....                   | 2-522 |
| <i>nfsAuthUnixPrompt()</i>    | – modify the NFS UNIX authentication parameters .....                | 2-522 |
| <i>nfsAuthUnixSet()</i>       | – set the NFS UNIX authentication parameters .....                   | 2-523 |
| <i>nfsAuthUnixShow()</i>      | – display the NFS UNIX authentication parameters .....               | 2-523 |
| <i>nfsDevInfoGet()</i>        | – read configuration information from the requested NFS device ..... | 2-524 |
| <i>nfsDevListGet()</i>        | – create list of all the NFS devices in the system .....             | 2-524 |
| <i>nfsDevShow()</i>           | – display the mounted NFS devices .....                              | 2-525 |
| <i>nfsdInit()</i>             | – initialize the NFS server .....                                    | 2-525 |
| <i>nfsDrv()</i>               | – install the NFS driver .....                                       | 2-526 |
| <i>nfsDrvNumGet()</i>         | – return the IO system driver number for the nfs driver .....        | 2-526 |
| <i>nfsdStatusGet()</i>        | – get the status of the NFS server .....                             | 2-527 |
| <i>nfsdStatusShow()</i>       | – show the status of the NFS server .....                            | 2-527 |
| <i>nfsExport()</i>            | – specify a file system to be NFS exported .....                     | 2-528 |
| <i>nfsExportShow()</i>        | – display the exported file systems of a remote host .....           | 2-528 |
| <i>nfsHelp()</i>              | – display the NFS help menu .....                                    | 2-529 |
| <i>nfsIdSet()</i>             | – set the ID number of the NFS UNIX authentication parameters .....  | 2-530 |
| <i>nfsMount()</i>             | – mount an NFS file system .....                                     | 2-530 |
| <i>nfsMountAll()</i>          | – mount all file systems exported by a specified host .....          | 2-531 |
| <i>nfsUnexport()</i>          | – remove a file system from the list of exported file systems .....  | 2-531 |
| <i>nfsUnmount()</i>           | – unmount an NFS device .....                                        | 2-532 |
| <i>nicEndLoad()</i>           | – initialize the driver and device .....                             | 2-532 |
| <i>nicEvbattach()</i>         | – publish and initialize the nicEvb network interface driver .....   | 2-533 |
| <i>nicEvbInitParse()</i>      | – parse the initialization string .....                              | 2-533 |
| <i>nicTxStartup()</i>         | – the driver’s actual output routine .....                           | 2-534 |
| <i>npc()</i>                  | – return the contents of the next program counter (SPARC) .....      | 2-534 |
| <i>ns16550DevInit()</i>       | – initialize an NS16550 channel .....                                | 2-535 |
| <i>ns16550Int()</i>           | – interrupt level processing .....                                   | 2-535 |
| <i>ns16550IntEx()</i>         | – miscellaneous interrupt processing .....                           | 2-536 |
| <i>ns16550IntRd()</i>         | – handle a receiver interrupt .....                                  | 2-536 |
| <i>ns16550IntWr()</i>         | – handle a transmitter interrupt .....                               | 2-537 |
| <i>ntInt()</i>                | – handle controller interrupt .....                                  | 2-537 |
| <i>ntLoad()</i>               | – initialize the driver and device .....                             | 2-538 |
| <i>ntMemInit()</i>            | – initialize memory for the chip .....                               | 2-538 |
| <i>ntParse()</i>              | – parse the init string .....                                        | 2-539 |
| <i>ntPassFsDevInit()</i>      | – associate a device with ntPassFs file system functions .....       | 2-539 |

|                             |                                                                        |       |
|-----------------------------|------------------------------------------------------------------------|-------|
| <i>ntPassFsInit()</i>       | – prepare to use the ntPassFs library .....                            | 2-540 |
| <i>ntPollStart()</i>        | – start polled mode operations .....                                   | 2-540 |
| <i>ntPollStop()</i>         | – stop polled mode operations .....                                    | 2-541 |
| <i>o0()</i>                 | – return the contents of register o0 (also o1 – o7) (SPARC) .....      | 2-541 |
| <i>open()</i>               | – open a file .....                                                    | 2-542 |
| <i>opendir()</i>            | – open a directory for searching (POSIX) .....                         | 2-543 |
| <i>operator delete()</i>    | – default run-time support for memory deallocation (C++) .....         | 2-543 |
| <i>operator new()</i>       | – default run-time support for operator new (C++) .....                | 2-544 |
| <i>operator new()</i>       | – default run-time support for operator new (nothrow) (C++) .....      | 2-544 |
| <i>operator new()</i>       | – run-time support for operator new with placement (C++) .....         | 2-545 |
| <i>ospfExtRouteAdd()</i>    | – import external route into OSPF domain (OSPF Opt.) .....             | 2-545 |
| <i>ospfExtRouteDelete()</i> | – delete external route imported into OSPF (OSPF Opt.) .....           | 2-546 |
| <i>ospfInit()</i>           | – function to initialize OSPF routing (OSPF Opt.) .....                | 2-546 |
| <i>ospfNbmaDstAdd()</i>     | – add NBMA destination .....                                           | 2-547 |
| <i>ospfNbmaDstDelete()</i>  | – delete NBMA destination .....                                        | 2-548 |
| <i>ospfTerminate()</i>      | – free OSPF resources and delete OSPF tasks .....                      | 2-548 |
| <i>passFsDevInit()</i>      | – associate a device with passFs file system functions .....           | 2-549 |
| <i>passFsInit()</i>         | – prepare to use the passFs library .....                              | 2-549 |
| <i>pause()</i>              | – suspend the task until delivery of a signal (POSIX) .....            | 2-550 |
| <i>pc()</i>                 | – return the contents of the program counter .....                     | 2-550 |
| <i>pccardAtaEnabler()</i>   | – enable the PCMCIA-ATA device .....                                   | 2-551 |
| <i>pccardEltEnabler()</i>   | – enable the PCMCIA Etherlink III card .....                           | 2-551 |
| <i>pccardMkfs()</i>         | – initialize a device and mount a DOS file system .....                | 2-552 |
| <i>pccardMount()</i>        | – mount a DOS file system .....                                        | 2-552 |
| <i>pccardSramEnabler()</i>  | – enable the PCMCIA-SRAM driver .....                                  | 2-553 |
| <i>pccardTffsEnabler()</i>  | – enable the PCMCIA-TFFS driver .....                                  | 2-553 |
| <i>pcicInit()</i>           | – initialize the PCIC chip .....                                       | 2-554 |
| <i>pcicShow()</i>           | – show all configurations of the PCIC chip .....                       | 2-554 |
| <i>pcmciaid()</i>           | – handle task-level PCMCIA events .....                                | 2-555 |
| <i>pcmciaInit()</i>         | – initialize the PCMCIA event-handling package .....                   | 2-555 |
| <i>pcmciaShow()</i>         | – show all configurations of the PCMCIA chip .....                     | 2-555 |
| <i>pcmciaShowInit()</i>     | – initialize all show routines for PCMCIA drivers .....                | 2-556 |
| <i>pcw()</i>                | – return the contents of the pcw register (i960) .....                 | 2-556 |
| <i>pentiumBtc()</i>         | – execute atomic compare-and-exchange instruction to clear a bit ..... | 2-557 |
| <i>pentiumBts()</i>         | – execute atomic compare-and-exchange instruction to set a bit .....   | 2-557 |
| <i>pentiumCr4Get()</i>      | – Get a content of CR4 register .....                                  | 2-558 |
| <i>pentiumCr4Set()</i>      | – Set a specified value to CR4 register .....                          | 2-558 |
| <i>pentiumMcaShow()</i>     | – show MCA (Machine Check Architecture) registers .....                | 2-558 |
| <i>pentiumMsrGet()</i>      | – get a content of the specified MSR (Model Specific Register) .....   | 2-559 |
| <i>pentiumMsrSet()</i>      | – set a value to the specified MSR (Model Specific Registers) .....    | 2-559 |
| <i>pentiumMtrrDisable()</i> | – disable MTRR (Memory Type Range Register) .....                      | 2-560 |
| <i>pentiumMtrrEnable()</i>  | – enable MTRR (Memory Type Range Register) .....                       | 2-560 |
| <i>pentiumMtrrGet()</i>     | – get MTRRs to a specified MTRR table .....                            | 2-560 |
| <i>pentiumMtrrSet()</i>     | – set MTRRs from specified MTRR table with WRMSR instruction. ....     | 2-561 |
| <i>pentiumPmcGet()</i>      | – get contents of PMC0 and PMC1 .....                                  | 2-561 |
| <i>pentiumPmcGet0()</i>     | – get a content of PMC0 .....                                          | 2-562 |
| <i>pentiumPmcGet1()</i>     | – get a content of PMC1 .....                                          | 2-562 |
| <i>pentiumPmcReset()</i>    | – reset both PMC0 and PMC1 .....                                       | 2-563 |

|                              |                                                                          |       |
|------------------------------|--------------------------------------------------------------------------|-------|
| <i>pentiumPmcReset0()</i>    | – reset PMC0 .....                                                       | 2-563 |
| <i>pentiumPmcReset1()</i>    | – reset PMC1 .....                                                       | 2-563 |
| <i>pentiumPmcShow()</i>      | – show PMCs (Performance Monitoring Counters) .....                      | 2-564 |
| <i>pentiumPmcStart()</i>     | – start both PMC0 and PMC1 .....                                         | 2-564 |
| <i>pentiumPmcStop()</i>      | – stop both PMC0 and PMC1 .....                                          | 2-565 |
| <i>pentiumPmcStop1()</i>     | – stop PMC1 .....                                                        | 2-565 |
| <i>pentiumSerialize()</i>    | – execute a serializing instruction CPUID .....                          | 2-565 |
| <i>pentiumTlbFlush()</i>     | – flush TLBs (Translation Lookaside Buffers) .....                       | 2-566 |
| <i>pentiumTscGet32()</i>     | – get a lower half of the 64Bit TSC (Timestamp Counter) .....            | 2-566 |
| <i>pentiumTscGet64()</i>     | – get 64Bit TSC (Timestamp Counter) .....                                | 2-567 |
| <i>pentiumTscReset()</i>     | – reset the TSC (Timestamp Counter) .....                                | 2-567 |
| <i>period()</i>              | – spawn a task to call a function periodically .....                     | 2-567 |
| <i>periodRun()</i>           | – call a function periodically .....                                     | 2-568 |
| <i>error()</i>               | – map an error number in errno to an error message (ANSI) .....          | 2-569 |
| <i>pfp()</i>                 | – return the contents of register pfp (i960) .....                       | 2-569 |
| <i>ping()</i>                | – test that a remote host is reachable .....                             | 2-570 |
| <i>pingLibInit()</i>         | – initialize the <i>ping()</i> utility .....                             | 2-571 |
| <i>pipeDevCreate()</i>       | – create a pipe device .....                                             | 2-571 |
| <i>pipeDrv()</i>             | – initialize the pipe driver .....                                       | 2-572 |
| <i>pow()</i>                 | – compute the value of a number raised to a specified power (ANSI) ..... | 2-572 |
| <i>powf()</i>                | – compute the value of a number raised to a specified power (ANSI) ..... | 2-573 |
| <i>ppc403DevInit()</i>       | – initialize the serial port unit .....                                  | 2-574 |
| <i>ppc403DummyCallback()</i> | – dummy callback routine .....                                           | 2-574 |
| <i>ppc403IntEx()</i>         | – handle error interrupts .....                                          | 2-575 |
| <i>ppc403IntRd()</i>         | – handle a receiver interrupt .....                                      | 2-575 |
| <i>ppc403IntWr()</i>         | – handle a transmitter interrupt .....                                   | 2-576 |
| <i>ppc860DevInit()</i>       | – initialize the SMC .....                                               | 2-576 |
| <i>ppc860Int()</i>           | – handle an SMC interrupt .....                                          | 2-576 |
| <i>pppDelete()</i>           | – delete a PPP network interface .....                                   | 2-577 |
| <i>pppHookAdd()</i>          | – add a hook routine on a unit basis .....                               | 2-577 |
| <i>pppHookDelete()</i>       | – delete a hook routine on a unit basis .....                            | 2-578 |
| <i>pppInfoGet()</i>          | – get PPP link status information .....                                  | 2-579 |
| <i>pppInfoShow()</i>         | – display PPP link status information .....                              | 2-579 |
| <i>pppInit()</i>             | – initialize a PPP network interface .....                               | 2-580 |
| <i>pppSecretAdd()</i>        | – add a secret to the PPP authentication secrets table .....             | 2-587 |
| <i>pppSecretDelete()</i>     | – delete a secret from the PPP authentication secrets table .....        | 2-588 |
| <i>pppSecretShow()</i>       | – display the PPP authentication secrets table .....                     | 2-589 |
| <i>pppstatGet()</i>          | – get PPP link statistics .....                                          | 2-589 |
| <i>pppstatShow()</i>         | – display PPP link statistics .....                                      | 2-590 |
| <i>printErr()</i>            | – write a formatted string to the standard error stream .....            | 2-590 |
| <i>printErrno()</i>          | – print the definition of a specified error status value .....           | 2-591 |
| <i>printf()</i>              | – write a formatted string to the standard output stream (ANSI) .....    | 2-591 |
| <i>printLogo()</i>           | – print the VxWorks logo .....                                           | 2-595 |
| <i>proxyArpLibInit()</i>     | – initialize proxy ARP .....                                             | 2-595 |
| <i>proxyNetCreate()</i>      | – create a proxy ARP network .....                                       | 2-596 |
| <i>proxyNetDelete()</i>      | – delete a proxy network .....                                           | 2-597 |
| <i>proxyNetShow()</i>        | – show proxy ARP networks .....                                          | 2-597 |
| <i>proxyPortFwdOff()</i>     | – disable broadcast forwarding for a particular port .....               | 2-598 |

|                              |                                                                            |       |
|------------------------------|----------------------------------------------------------------------------|-------|
| <i>proxyPortFwdOn()</i>      | – enable broadcast forwarding for a particular port .....                  | 2-598 |
| <i>proxyPortShow()</i>       | – show enabled ports .....                                                 | 2-598 |
| <i>proxyReg()</i>            | – register a proxy client .....                                            | 2-599 |
| <i>proxyUnreg()</i>          | – unregister a proxy client .....                                          | 2-599 |
| <i>psr()</i>                 | – return the contents of the processor status register (SPARC) .....       | 2-600 |
| <i>psrShow()</i>             | – display the meaning of a specified psr value, symbolically (SPARC) ..... | 2-600 |
| <i>psrShow()</i>             | – display the meaning of a specified PSR value, symbolically (ARM) .....   | 2-601 |
| <i>ptyDevCreate()</i>        | – create a pseudo terminal .....                                           | 2-602 |
| <i>ptyDrv()</i>              | – initialize the pseudo-terminal driver .....                              | 2-602 |
| <i>putc()</i>                | – write a character to a stream (ANSI) .....                               | 2-603 |
| <i>putchar()</i>             | – write a character to the standard output stream (ANSI) .....             | 2-603 |
| <i>putenv()</i>              | – set an environment variable .....                                        | 2-604 |
| <i>puts()</i>                | – write a string to the standard output stream (ANSI) .....                | 2-604 |
| <i>putw()</i>                | – write a word (32-bit integer) to a stream .....                          | 2-605 |
| <i>pwd()</i>                 | – print the current default directory .....                                | 2-605 |
| <i>qsort()</i>               | – sort an array of objects (ANSI) .....                                    | 2-606 |
| <i>r0()</i>                  | – return the contents of register r0 (also r1 – r14) (ARM) .....           | 2-606 |
| <i>r3()</i>                  | – return the contents of register r3 (also r4 – r15) (i960) .....          | 2-607 |
| <i>raise()</i>               | – send a signal to the caller’s task .....                                 | 2-607 |
| <i>ramDevCreate()</i>        | – create a RAM disk device .....                                           | 2-608 |
| <i>ramDrv()</i>              | – prepare a RAM disk driver for use (optional) .....                       | 2-609 |
| <i>rand()</i>                | – generate a pseudo-random integer between 0 and RAND_MAX (ANSI) .....     | 2-610 |
| <i>rawFsDevInit()</i>        | – associate a block device with raw volume functions .....                 | 2-610 |
| <i>rawFsInit()</i>           | – prepare to use the raw volume library .....                              | 2-611 |
| <i>rawFsModeChange()</i>     | – modify the mode of a raw device volume .....                             | 2-611 |
| <i>rawFsReadyChange()</i>    | – notify rawFsLib of a change in ready status .....                        | 2-612 |
| <i>rawFsVolUnmount()</i>     | – disable a raw device volume .....                                        | 2-613 |
| <i>rcmd()</i>                | – execute a shell command on a remote machine .....                        | 2-613 |
| <i>read()</i>                | – read bytes from a file or device .....                                   | 2-614 |
| <i>readdir()</i>             | – read one entry from a directory (POSIX) .....                            | 2-614 |
| <i>realloc()</i>             | – reallocate a block of memory (ANSI) .....                                | 2-615 |
| <i>reboot()</i>              | – reset network devices and transfer control to boot ROMs .....            | 2-616 |
| <i>rebootHookAdd()</i>       | – add a routine to be called at reboot .....                               | 2-616 |
| <i>recv()</i>                | – receive data from a socket .....                                         | 2-617 |
| <i>recvfrom()</i>            | – receive a message from a socket .....                                    | 2-618 |
| <i>recvmsg()</i>             | – receive a message from a socket .....                                    | 2-618 |
| <i>reld()</i>                | – reload an object module .....                                            | 2-619 |
| <i>remCurIdGet()</i>         | – get the current user name and password .....                             | 2-620 |
| <i>remCurIdSet()</i>         | – set the remote user name and password .....                              | 2-620 |
| <i>remove()</i>              | – remove a file (ANSI) .....                                               | 2-621 |
| <i>rename()</i>              | – change the name of a file .....                                          | 2-621 |
| <i>repeat()</i>              | – spawn a task to call a function repeatedly .....                         | 2-622 |
| <i>repeatRun()</i>           | – call a function repeatedly .....                                         | 2-622 |
| <i>resolvDNComp()</i>        | – compress a DNS name in a DNS packet .....                                | 2-623 |
| <i>resolvDNExpand()</i>      | – expand a DNS compressed name from a DNS packet .....                     | 2-624 |
| <i>resolvGetHostByAddr()</i> | – query the DNS server for the host name of an IP address .....            | 2-624 |
| <i>resolvGetHostByName()</i> | – query the DNS server for the IP address of a host .....                  | 2-625 |
| <i>resolvInit()</i>          | – initialize the resolver library .....                                    | 2-626 |

|                                |                                                                           |       |
|--------------------------------|---------------------------------------------------------------------------|-------|
| <i>resolvMkQuery()</i>         | – create all types of DNS queries .....                                   | 2-627 |
| <i>resolvParamsGet()</i>       | – get the parameters which control the resolver library .....             | 2-628 |
| <i>resolvParamsSet()</i>       | – set the parameters which control the resolver library .....             | 2-628 |
| <i>resolvQuery()</i>           | – construct a query, send it, wait for a response .....                   | 2-629 |
| <i>resolvSend()</i>            | – send a pre-formatted query and return the answer .....                  | 2-630 |
| <i>rewind()</i>                | – set the file position indicator to the beginning of a file (ANSI) ..... | 2-631 |
| <i>rewinddir()</i>             | – reset position to the start of a directory (POSIX) .....                | 2-631 |
| <i>rindex()</i>                | – find the last occurrence of a character in a string .....               | 2-632 |
| <i>rip()</i>                   | – return the contents of register rip (i960) .....                        | 2-632 |
| <i>ripAuthHook()</i>           | – sample authentication hook .....                                        | 2-633 |
| <i>ripAuthHookAdd()</i>        | – add an authentication hook to a RIP interface .....                     | 2-633 |
| <i>ripAuthHookDelete()</i>     | – remove an authentication hook from a RIP interface .....                | 2-636 |
| <i>ripDebugLevelSet()</i>      | – specify amount of debugging output .....                                | 2-636 |
| <i>ripFilterDisable()</i>      | – prevent strict border gateway filtering .....                           | 2-637 |
| <i>ripFilterEnable()</i>       | – activate strict border gateway filtering .....                          | 2-637 |
| <i>ripIfReset()</i>            | – alter the RIP configuration after an interface changes .....            | 2-638 |
| <i>ripIfSearch()</i>           | – add new interfaces to the internal list .....                           | 2-638 |
| <i>ripLeakHookAdd()</i>        | – add a hook to bypass the RIP and kernel routing tables .....            | 2-639 |
| <i>ripLeakHookDelete()</i>     | – remove a table bypass hook from a RIP interface .....                   | 2-639 |
| <i>ripLibInit()</i>            | – initialize the RIP routing library .....                                | 2-640 |
| <i>ripRouteShow()</i>          | – display the internal routing table maintained by RIP .....              | 2-641 |
| <i>ripSendHookAdd()</i>        | – add an update filter to a RIP interface .....                           | 2-642 |
| <i>ripSendHookDelete()</i>     | – remove an update filter from a RIP interface .....                      | 2-643 |
| <i>ripShutdown()</i>           | – terminate all RIP processing .....                                      | 2-643 |
| <i>rlogin()</i>                | – log in to a remote host .....                                           | 2-644 |
| <i>rlogind()</i>               | – the VxWorks remote login daemon .....                                   | 2-644 |
| <i>rlogInit()</i>              | – initialize the remote login facility .....                              | 2-645 |
| <i>rm()</i>                    | – remove a file .....                                                     | 2-645 |
| <i>rmdir()</i>                 | – remove a directory .....                                                | 2-646 |
| <i>rngBufGet()</i>             | – get characters from a ring buffer .....                                 | 2-646 |
| <i>rngBufPut()</i>             | – put bytes into a ring buffer .....                                      | 2-647 |
| <i>rngCreate()</i>             | – create an empty ring buffer .....                                       | 2-647 |
| <i>rngDelete()</i>             | – delete a ring buffer .....                                              | 2-648 |
| <i>rngFlush()</i>              | – make a ring buffer empty .....                                          | 2-648 |
| <i>rngFreeBytes()</i>          | – determine the number of free bytes in a ring buffer .....               | 2-649 |
| <i>rngIsEmpty()</i>            | – test if a ring buffer is empty .....                                    | 2-649 |
| <i>rngIsFull()</i>             | – test if a ring buffer is full (no more room) .....                      | 2-650 |
| <i>rngMoveAhead()</i>          | – advance a ring pointer by n bytes .....                                 | 2-650 |
| <i>rngNBytes()</i>             | – determine the number of bytes in a ring buffer .....                    | 2-651 |
| <i>rngPutAhead()</i>           | – put a byte ahead in a ring buffer without moving ring pointers .....    | 2-651 |
| <i>romStart()</i>              | – generic ROM initialization .....                                        | 2-652 |
| <i>round()</i>                 | – round a number to the nearest integer .....                             | 2-652 |
| <i>roundf()</i>                | – round a number to the nearest integer .....                             | 2-653 |
| <i>routeAdd()</i>              | – add a route .....                                                       | 2-653 |
| <i>routeDelete()</i>           | – delete a route .....                                                    | 2-654 |
| <i>routeNetAdd()</i>           | – add a route to a destination that is a network .....                    | 2-655 |
| <i>routeProtoPrioritySet()</i> | – set the priority of routes added by the routing protocol .....          | 2-655 |
| <i>routeShow()</i>             | – display host and network routing tables .....                           | 2-656 |

|                                 |                                                                           |       |
|---------------------------------|---------------------------------------------------------------------------|-------|
| <i>routeStatShow()</i>          | – display routing statistics .....                                        | 2-657 |
| <i>rpcInit()</i>                | – initialize the RPC package .....                                        | 2-657 |
| <i>rpcTaskInit()</i>            | – initialize a task’s access to the RPC package .....                     | 2-658 |
| <i>rresvport()</i>              | – open a socket with a privileged port bound to it .....                  | 2-658 |
| <i>rt11FsDateSet()</i>          | – set the rt11Fs file system date .....                                   | 2-658 |
| <i>rt11FsDevInit()</i>          | – initialize the rt11Fs device descriptor .....                           | 2-659 |
| <i>rt11FsInit()</i>             | – prepare to use the rt11Fs library .....                                 | 2-660 |
| <i>rt11FsMkfs()</i>             | – initialize a device and create an rt11Fs file system .....              | 2-661 |
| <i>rt11FsModeChange()</i>       | – modify the mode of an rt11Fs volume .....                               | 2-661 |
| <i>rt11FsReadyChange()</i>      | – notify rt11Fs of a change in ready status .....                         | 2-662 |
| <i>s()</i>                      | – single-step a task .....                                                | 2-662 |
| <i>sa1100DevInit()</i>          | – initialise an SA1100 channel .....                                      | 2-663 |
| <i>sa1100Int()</i>              | – handle an interrupt .....                                               | 2-664 |
| <i>saIoWrite()</i>              | – send a packet to the master agent’s message queue .....                 | 2-664 |
| <i>saIpcFree()</i>              | – free the specified IPC mechanism .....                                  | 2-665 |
| <i>saMsgBuild()</i>             | – build and encode a message and send it to the master agent .....        | 2-665 |
| <i>scanf()</i>                  | – read and convert characters from the standard input stream (ANSI) ..... | 2-666 |
| <i>sched_get_priority_max()</i> | – get the maximum priority (POSIX) .....                                  | 2-667 |
| <i>sched_get_priority_min()</i> | – get the minimum priority (POSIX) .....                                  | 2-667 |
| <i>sched_getparam()</i>         | – get the scheduling parameters for a specified task (POSIX) .....        | 2-668 |
| <i>sched_getscheduler()</i>     | – get the current scheduling policy (POSIX) .....                         | 2-669 |
| <i>sched_rr_get_interval()</i>  | – get the current time slice (POSIX) .....                                | 2-669 |
| <i>sched_setparam()</i>         | – set a task’s priority (POSIX) .....                                     | 2-670 |
| <i>sched_setscheduler()</i>     | – set scheduling policy and scheduling parameters (POSIX) .....           | 2-671 |
| <i>sched_yield()</i>            | – relinquish the CPU (POSIX) .....                                        | 2-671 |
| <i>scsi2IfInit()</i>            | – initialize the SCSI-2 interface to <b>scsiLib</b> .....                 | 2-672 |
| <i>scsiAutoConfig()</i>         | – configure all devices connected to a SCSI controller .....              | 2-672 |
| <i>scsiBlkDevCreate()</i>       | – define a logical partition on a SCSI block device .....                 | 2-673 |
| <i>scsiBlkDevInit()</i>         | – initialize fields in a SCSI logical partition .....                     | 2-673 |
| <i>scsiBlkDevShow()</i>         | – show the <b>BLK_DEV</b> structures on a specified physical device ..... | 2-674 |
| <i>scsiBusReset()</i>           | – pulse the reset signal on the SCSI bus .....                            | 2-674 |
| <i>scsiCacheSnoopDisable()</i>  | – inform SCSI that hardware snooping of caches is disabled .....          | 2-675 |
| <i>scsiCacheSnoopEnable()</i>   | – inform SCSI that hardware snooping of caches is enabled .....           | 2-675 |
| <i>scsiCacheSynchronize()</i>   | – synchronize the caches for data coherency .....                         | 2-676 |
| <i>scsiErase()</i>              | – issue an ERASE command to a SCSI device .....                           | 2-677 |
| <i>scsiFormatUnit()</i>         | – issue a <b>FORMAT_UNIT</b> command to a SCSI device .....               | 2-677 |
| <i>scsiIdentMsgBuild()</i>      | – build an identification message .....                                   | 2-678 |
| <i>scsiIdentMsgParse()</i>      | – parse an identification message .....                                   | 2-678 |
| <i>scsiInquiry()</i>            | – issue an INQUIRY command to a SCSI device .....                         | 2-679 |
| <i>scsiIoctl()</i>              | – perform a device-specific I/O control function .....                    | 2-680 |
| <i>scsiLoadUnit()</i>           | – issue a <b>LOAD/UNLOAD</b> command to a SCSI device .....               | 2-680 |
| <i>scsiMgrBusReset()</i>        | – handle a controller-bus reset event .....                               | 2-681 |
| <i>scsiMgrCtrlEvent()</i>       | – send an event to the SCSI controller state machine .....                | 2-681 |
| <i>scsiMgrEventNotify()</i>     | – notify the SCSI manager of a SCSI (controller) event .....              | 2-682 |
| <i>scsiMgrShow()</i>            | – show status information for the SCSI manager .....                      | 2-682 |
| <i>scsiMgrThreadEvent()</i>     | – send an event to the thread state machine .....                         | 2-683 |
| <i>scsiModeSelect()</i>         | – issue a <b>MODE_SELECT</b> command to a SCSI device .....               | 2-684 |
| <i>scsiModeSense()</i>          | – issue a <b>MODE_SENSE</b> command to a SCSI device .....                | 2-684 |

|                                 |                                                                                |       |
|---------------------------------|--------------------------------------------------------------------------------|-------|
| <i>scsiMsgInComplete()</i>      | – handle a complete SCSI message received from the target .....                | 2-685 |
| <i>scsiMsgOutComplete()</i>     | – perform post-processing after a SCSI message is sent .....                   | 2-685 |
| <i>scsiMsgOutReject()</i>       | – perform post-processing when an outgoing message is rejected .....           | 2-686 |
| <i>scsiPhysDevCreate()</i>      | – create a SCSI physical device structure .....                                | 2-686 |
| <i>scsiPhysDevDelete()</i>      | – delete a SCSI physical-device structure .....                                | 2-687 |
| <i>scsiPhysDevIdGet()</i>       | – return a pointer to a <b>SCSI_PHYS_DEV</b> structure .....                   | 2-687 |
| <i>scsiPhysDevShow()</i>        | – show status information for a physical device .....                          | 2-688 |
| <i>scsiRdSecs()</i>             | – read sector(s) from a SCSI block device .....                                | 2-688 |
| <i>scsiRdTape()</i>             | – read bytes or blocks from a SCSI tape device .....                           | 2-689 |
| <i>scsiReadCapacity()</i>       | – issue a <b>READ_CAPACITY</b> command to a SCSI device .....                  | 2-689 |
| <i>scsiRelease()</i>            | – issue a <b>RELEASE</b> command to a SCSI device .....                        | 2-690 |
| <i>scsiReleaseUnit()</i>        | – issue a <b>RELEASE UNIT</b> command to a SCSI device .....                   | 2-690 |
| <i>scsiReqSense()</i>           | – issue a <b>REQUEST_SENSE</b> command to a SCSI device and read results ..... | 2-691 |
| <i>scsiReserve()</i>            | – issue a <b>RESERVE</b> command to a SCSI device .....                        | 2-691 |
| <i>scsiReserveUnit()</i>        | – issue a <b>RESERVE UNIT</b> command to a SCSI device .....                   | 2-692 |
| <i>scsiRewind()</i>             | – issue a <b>REWIND</b> command to a SCSI device .....                         | 2-692 |
| <i>scsiSeqDevCreate()</i>       | – create a SCSI sequential device .....                                        | 2-693 |
| <i>scsiSeqIoctl()</i>           | – perform an I/O control function for sequential access devices .....          | 2-694 |
| <i>scsiSeqReadBlockLimits()</i> | – issue a <b>READ_BLOCK_LIMITS</b> command to a SCSI device .....              | 2-694 |
| <i>scsiSeqStatusCheck()</i>     | – detect a change in media .....                                               | 2-695 |
| <i>scsiShow()</i>               | – list the physical devices attached to a SCSI controller .....                | 2-695 |
| <i>scsiSpace()</i>              | – move the tape on a specified physical SCSI device .....                      | 2-696 |
| <i>scsiStartStopUnit()</i>      | – issue a <b>START_STOP_UNIT</b> command to a SCSI device .....                | 2-696 |
| <i>scsiSyncXferNegotiate()</i>  | – initiate or continue negotiating transfer parameters .....                   | 2-697 |
| <i>scsiTapeModeSelect()</i>     | – issue a <b>MODE_SELECT</b> command to a SCSI tape device .....               | 2-697 |
| <i>scsiTapeModeSense()</i>      | – issue a <b>MODE_SENSE</b> command to a SCSI tape device .....                | 2-698 |
| <i>scsiTargetOptionsGet()</i>   | – get options for one or all SCSI targets .....                                | 2-698 |
| <i>scsiTargetOptionsSet()</i>   | – set options for one or all SCSI targets .....                                | 2-699 |
| <i>scsiTestUnitRdy()</i>        | – issue a <b>TEST_UNIT_READY</b> command to a SCSI device .....                | 2-700 |
| <i>scsiThreadInit()</i>         | – perform generic SCSI thread initialization .....                             | 2-700 |
| <i>scsiWideXferNegotiate()</i>  | – initiate or continue negotiating wide parameters .....                       | 2-701 |
| <i>scsiWrtFileMarks()</i>       | – write file marks to a SCSI sequential device .....                           | 2-701 |
| <i>scsiWrtSecs()</i>            | – write sector(s) to a SCSI block device .....                                 | 2-702 |
| <i>scsiWrtTape()</i>            | – write data to a SCSI tape device .....                                       | 2-702 |
| <i>select()</i>                 | – pend on a set of file descriptors .....                                      | 2-703 |
| <i>selectInit()</i>             | – initialize the select facility .....                                         | 2-704 |
| <i>selNodeAdd()</i>             | – add a wake-up node to a <i>select()</i> wake-up list .....                   | 2-704 |
| <i>selNodeDelete()</i>          | – find and delete a node from a <i>select()</i> wake-up list .....             | 2-705 |
| <i>selWakeup()</i>              | – wake up a task pending in <i>select()</i> .....                              | 2-705 |
| <i>selWakeupAll()</i>           | – wake up all tasks in a <i>select()</i> wake-up list .....                    | 2-706 |
| <i>selWakeupListInit()</i>      | – initialize a <i>select()</i> wake-up list .....                              | 2-706 |
| <i>selWakeupListLen()</i>       | – get the number of nodes in a <i>select()</i> wake-up list .....              | 2-707 |
| <i>selWakeupType()</i>          | – get the type of a <i>select()</i> wake-up node .....                         | 2-707 |
| <i>semBCreate()</i>             | – create and initialize a binary semaphore .....                               | 2-708 |
| <i>semBSmCreate()</i>           | – create and initialize a shared memory binary semaphore (VxMP Opt.) .....     | 2-708 |
| <i>semCCreate()</i>             | – create and initialize a counting semaphore .....                             | 2-709 |
| <i>semClear()</i>               | – take a release 4.x semaphore, if the semaphore is available .....            | 2-710 |
| <i>semCreate()</i>              | – create and initialize a release 4.x binary semaphore .....                   | 2-710 |

|                           |                                                                              |       |
|---------------------------|------------------------------------------------------------------------------|-------|
| <i>semCsmCreate()</i>     | – create and initialize a shared memory counting semaphore (VxMP Opt.) ..... | 2-710 |
| <i>semDelete()</i>        | – delete a semaphore .....                                                   | 2-711 |
| <i>semFlush()</i>         | – unblock every task pended on a semaphore .....                             | 2-712 |
| <i>semGive()</i>          | – give a semaphore .....                                                     | 2-712 |
| <i>semInfo()</i>          | – get a list of task IDs that are blocked on a semaphore .....               | 2-713 |
| <i>semInit()</i>          | – initialize a static binary semaphore .....                                 | 2-714 |
| <i>semMCreate()</i>       | – create and initialize a mutual-exclusion semaphore .....                   | 2-714 |
| <i>semMGiveForce()</i>    | – give a mutual-exclusion semaphore without restrictions .....               | 2-715 |
| <i>semPxLibInit()</i>     | – initialize POSIX semaphore support .....                                   | 2-716 |
| <i>semPxShowInit()</i>    | – initialize the POSIX semaphore show facility .....                         | 2-716 |
| <i>semShow()</i>          | – show information about a semaphore .....                                   | 2-717 |
| <i>semShowInit()</i>      | – initialize the semaphore show facility .....                               | 2-717 |
| <i>semTake()</i>          | – take a semaphore .....                                                     | 2-718 |
| <i>sem_close()</i>        | – close a named semaphore (POSIX) .....                                      | 2-719 |
| <i>sem_destroy()</i>      | – destroy an unnamed semaphore (POSIX) .....                                 | 2-719 |
| <i>sem_getvalue()</i>     | – get the value of a semaphore (POSIX) .....                                 | 2-720 |
| <i>sem_init()</i>         | – initialize an unnamed semaphore (POSIX) .....                              | 2-721 |
| <i>sem_open()</i>         | – initialize/open a named semaphore (POSIX) .....                            | 2-721 |
| <i>sem_post()</i>         | – unlock (give) a semaphore (POSIX) .....                                    | 2-723 |
| <i>sem_trywait()</i>      | – lock (take) a semaphore, returning error if unavailable (POSIX) .....      | 2-723 |
| <i>sem_unlink()</i>       | – remove a named semaphore (POSIX) .....                                     | 2-724 |
| <i>sem_wait()</i>         | – lock (take) a semaphore, blocking if not available (POSIX) .....           | 2-725 |
| <i>send()</i>             | – send data to a socket .....                                                | 2-725 |
| <i>sendmsg()</i>          | – send a message to a socket .....                                           | 2-726 |
| <i>sendto()</i>           | – send a message to a socket .....                                           | 2-727 |
| <i>set_new_handler()</i>  | – set <i>new_handler</i> to user-defined function (C++) .....                | 2-727 |
| <i>set_terminate()</i>    | – set <i>terminate</i> to user-defined function (C++) .....                  | 2-728 |
| <i>setbuf()</i>           | – specify the buffering for a stream (ANSI) .....                            | 2-728 |
| <i>setbuffer()</i>        | – specify buffering for a stream .....                                       | 2-729 |
| <i>sethostname()</i>      | – set the symbolic name of this machine .....                                | 2-729 |
| <i>setjmp()</i>           | – save the calling environment in a <i>jmp_buf</i> argument (ANSI) .....     | 2-730 |
| <i>setlinebuf()</i>       | – set line buffering for standard output or standard error .....             | 2-731 |
| <i>setlocale()</i>        | – set the appropriate locale (ANSI) .....                                    | 2-731 |
| <i>setsockopt()</i>       | – set socket options .....                                                   | 2-732 |
| <i>setvbuf()</i>          | – specify buffering for a stream (ANSI) .....                                | 2-737 |
| <i>shell()</i>            | – the shell entry point .....                                                | 2-738 |
| <i>shellHistory()</i>     | – display or set the size of shell history .....                             | 2-739 |
| <i>shellInit()</i>        | – start the shell .....                                                      | 2-739 |
| <i>shellLock()</i>        | – lock access to the shell .....                                             | 2-740 |
| <i>shellOrigStdSet()</i>  | – set the shell's default input/output/error file descriptors .....          | 2-740 |
| <i>shellPromptSet()</i>   | – change the shell prompt .....                                              | 2-741 |
| <i>shellScriptAbort()</i> | – signal the shell to stop processing a script .....                         | 2-741 |
| <i>show()</i>             | – print information on a specified object .....                              | 2-741 |
| <i>shutdown()</i>         | – shut down a network connection .....                                       | 2-742 |
| <i>sigaction()</i>        | – examine and/or specify the action associated with a signal (POSIX) .....   | 2-742 |
| <i>sigaddset()</i>        | – add a signal to a signal set (POSIX) .....                                 | 2-743 |
| <i>sigblock()</i>         | – add to a set of blocked signals .....                                      | 2-743 |
| <i>sigdelset()</i>        | – delete a signal from a signal set (POSIX) .....                            | 2-744 |



|                             |                                                                                   |       |
|-----------------------------|-----------------------------------------------------------------------------------|-------|
| <i>sigemptyset()</i>        | – initialize a signal set with no signals included (POSIX) .....                  | 2-744 |
| <i>sigfillset()</i>         | – initialize a signal set with all signals included (POSIX) .....                 | 2-745 |
| <i>sigInit()</i>            | – initialize the signal facilities .....                                          | 2-745 |
| <i>sigismember()</i>        | – test to see if a signal is in a signal set (POSIX) .....                        | 2-746 |
| <i>signal()</i>             | – specify the handler associated with a signal .....                              | 2-746 |
| <i>sigpending()</i>         | – retrieve the set of pending signals blocked from delivery (POSIX) .....         | 2-747 |
| <i>sigprocmask()</i>        | – examine and/or change the signal mask (POSIX) .....                             | 2-747 |
| <i>sigqueue()</i>           | – send a queued signal to a task .....                                            | 2-748 |
| <i>sigqueueInit()</i>       | – initialize the queued signal facilities .....                                   | 2-748 |
| <i>sigsetmask()</i>         | – set the signal mask .....                                                       | 2-749 |
| <i>sigsuspend()</i>         | – suspend the task until delivery of a signal (POSIX) .....                       | 2-749 |
| <i>sigtimedwait()</i>       | – wait for a signal .....                                                         | 2-750 |
| <i>sigvec()</i>             | – install a signal handler .....                                                  | 2-751 |
| <i>sigwaitinfo()</i>        | – wait for real-time signals .....                                                | 2-752 |
| <i>sin()</i>                | – compute a sine (ANSI) .....                                                     | 2-753 |
| <i>sincos()</i>             | – compute both a sine and cosine .....                                            | 2-753 |
| <i>sincosf()</i>            | – compute both a sine and cosine .....                                            | 2-754 |
| <i>sinf()</i>               | – compute a sine (ANSI) .....                                                     | 2-754 |
| <i>sinh()</i>               | – compute a hyperbolic sine (ANSI) .....                                          | 2-755 |
| <i>sinhf()</i>              | – compute a hyperbolic sine (ANSI) .....                                          | 2-755 |
| <i>slattach()</i>           | – publish the sl network interface and initialize the driver and device .....     | 2-756 |
| <i>slipBaudSet()</i>        | – set the baud rate for a SLIP interface .....                                    | 2-756 |
| <i>slipDelete()</i>         | – delete a SLIP interface .....                                                   | 2-757 |
| <i>slipInit()</i>           | – initialize a SLIP interface .....                                               | 2-757 |
| <i>smIfAttach()</i>         | – publish the sm interface and initialize the driver and device .....             | 2-758 |
| <i>smMemAddToPool()</i>     | – add memory to the shared memory system partition (VxMP Opt.) .....              | 2-759 |
| <i>smMemCalloc()</i>        | – allocate memory for array from shared memory system partition (VxMP Opt.) ..... | 2-760 |
| <i>smMemFindMax()</i>       | – find largest free block in the shared memory system partition (VxMP Opt.) ....  | 2-760 |
| <i>smMemFree()</i>          | – free a shared memory system partition block of memory (VxMP Opt.) .....         | 2-761 |
| <i>smMemMalloc()</i>        | – allocate block of memory from shared memory system partition (VxMP Opt.) .....  | 2-762 |
| <i>smMemOptionsSet()</i>    | – set the debug options for the shared memory system partition (VxMP Opt.) ...    | 2-762 |
| <i>smMemRealloc()</i>       | – reallocate block of memory from shared mem system partition (VxMP Opt.) .       | 2-763 |
| <i>smMemShow()</i>          | – show the shared memory system partition blocks and statistics (VxMP Opt.) .     | 2-764 |
| <i>smNameAdd()</i>          | – add a name to the shared memory name database (VxMP Opt.) .....                 | 2-765 |
| <i>smNameFind()</i>         | – look up a shared memory object by name (VxMP Opt.) .....                        | 2-766 |
| <i>smNameFindByValue()</i>  | – look up a shared memory object by value (VxMP Opt.) .....                       | 2-767 |
| <i>smNameRemove()</i>       | – remove object from the shared memory objects name database (VxMP Opt.) ..       | 2-768 |
| <i>smNameShow()</i>         | – show the contents of the shared memory objects name database (VxMP Opt.) ..     | 2-768 |
| <i>smNetAttach()</i>        | – attach the shared memory network interface .....                                | 2-769 |
| <i>smNetInetGet()</i>       | – get an address associated with a shared memory network interface .....          | 2-770 |
| <i>smNetInit()</i>          | – initialize the shared memory network driver .....                               | 2-771 |
| <i>smNetShow()</i>          | – show information about a shared memory network .....                            | 2-772 |
| <i>smObjAttach()</i>        | – attach the calling CPU to the shared memory objects facility (VxMP Opt.) .....  | 2-772 |
| <i>smObjGlobalToLocal()</i> | – convert a global address to a local address (VxMP Opt.) .....                   | 2-773 |
| <i>smObjInit()</i>          | – initialize a shared memory objects descriptor (VxMP Opt.) .....                 | 2-774 |
| <i>smObjLibInit()</i>       | – install the shared memory objects facility (VxMP Opt.) .....                    | 2-775 |
| <i>smObjLocalToGlobal()</i> | – convert a local address to a global address (VxMP Opt.) .....                   | 2-776 |
| <i>smObjSetup()</i>         | – initialize the shared memory objects facility (VxMP Opt.) .....                 | 2-776 |

|                                 |                                                                                 |       |
|---------------------------------|---------------------------------------------------------------------------------|-------|
| <i>smObjShow()</i>              | – display the current status of shared memory objects (VxMP Opt.) .....         | 2-777 |
| <i>smObjTimeoutLogEnable()</i>  | – enable/disable logging of failed attempts to take spin-lock (VxMP Opt.) ..... | 2-778 |
| <i>sn83932EndLoad()</i>         | – initialize the driver and device .....                                        | 2-779 |
| <i>snattach()</i>               | – publish the sn network interface and initialize the driver and device .....   | 2-779 |
| <i>snmpMasterCleanup()</i>      | – free up resources after a query times out .....                               | 2-780 |
| <i>snmpMasterHandlerAsync()</i> | – process messages from the subagent asynchronously .....                       | 2-780 |
| <i>snmpMasterHandlerWR()</i>    | – synchronous version of <i>snmpMasterHandlerAsync()</i> .....                  | 2-783 |
| <i>snmpMasterQueryHandler()</i> | – handles replies from the subagent .....                                       | 2-786 |
| <i>snmpMonitorSpawn()</i>       | – spawn tMonQue to run <i>snmpQueMonitor()</i> .....                            | 2-787 |
| <i>snmpSaHandlerAsync()</i>     | – asynchronous message processing routine for the subagent .....                | 2-788 |
| <i>snmpSaHandlerCleanup()</i>   | – cleanup routine for subagent .....                                            | 2-790 |
| <i>snmpSaHandlerContinue()</i>  | – subagent continuation function .....                                          | 2-791 |
| <i>snmpSaHandlerFinish()</i>    | – encode packet for subagent IO completion .....                                | 2-791 |
| <i>snmpSaHandlerWR()</i>        | – provide <i>snmpSaHandlerAsync()</i> functionality synchronously .....         | 2-792 |
| <i>snmpSaInit()</i>             | – initialize the subagent .....                                                 | 2-793 |
| <i>snmpSubEncode()</i>          | – encode a packet for transmission to master agent or subagent .....            | 2-794 |
| <i>sntpcTimeGet()</i>           | – retrieve the current time from a remote source .....                          | 2-795 |
| <i>sntpsClockSet()</i>          | – assign a routine to access the reference clock .....                          | 2-796 |
| <i>sntpsConfigSet()</i>         | – change SNTP server broadcast settings .....                                   | 2-797 |
| <i>sntpsNsecToFraction()</i>    | – convert portions of a second to NTP format .....                              | 2-798 |
| <i>so()</i>                     | – single-step, but step over a subroutine .....                                 | 2-798 |
| <i>socket()</i>                 | – open a socket .....                                                           | 2-799 |
| <i>sp()</i>                     | – spawn a task with default parameters .....                                    | 2-799 |
| <i>sprintf()</i>                | – write a formatted string to a buffer (ANSI) .....                             | 2-800 |
| <i>spy()</i>                    | – begin periodic task activity reports .....                                    | 2-801 |
| <i>spyClkStart()</i>            | – start collecting task activity data .....                                     | 2-801 |
| <i>spyClkStop()</i>             | – stop collecting task activity data .....                                      | 2-802 |
| <i>spyHelp()</i>                | – display task monitoring help menu .....                                       | 2-802 |
| <i>spyLibInit()</i>             | – initialize task cpu utilization tool package .....                            | 2-803 |
| <i>spyReport()</i>              | – display task activity data .....                                              | 2-803 |
| <i>spyStop()</i>                | – stop spying and reporting .....                                               | 2-804 |
| <i>spyTask()</i>                | – run periodic task activity reports .....                                      | 2-804 |
| <i>sqrt()</i>                   | – compute a non-negative square root (ANSI) .....                               | 2-804 |
| <i>sqrtf()</i>                  | – compute a non-negative square root (ANSI) .....                               | 2-805 |
| <i>squeeze()</i>                | – reclaim fragmented free space on an RT-11 volume .....                        | 2-805 |
| <i>sr()</i>                     | – return the contents of the status register (MC680x0) .....                    | 2-806 |
| <i>sramDevCreate()</i>          | – create a PCMCIA memory disk device .....                                      | 2-806 |
| <i>sramDrv()</i>                | – install a PCMCIA SRAM memory driver .....                                     | 2-807 |
| <i>sramMap()</i>                | – map PCMCIA memory onto a specified ISA address space .....                    | 2-807 |
| <i>srand()</i>                  | – reset the value of the seed used to generate random numbers (ANSI) .....      | 2-808 |
| <i>sscanf()</i>                 | – read and convert characters from an ASCII string (ANSI) .....                 | 2-808 |
| <i>st16552DevInit()</i>         | – initialise an ST16552 channel .....                                           | 2-812 |
| <i>st16552Int()</i>             | – interrupt level processing .....                                              | 2-812 |
| <i>st16552IntEx()</i>           | – miscellaneous interrupt processing .....                                      | 2-813 |
| <i>st16552IntRd()</i>           | – handle a receiver interrupt .....                                             | 2-813 |
| <i>st16552IntWr()</i>           | – handle a transmitter interrupt .....                                          | 2-814 |
| <i>st16552MuxInt()</i>          | – multiplexed interrupt level processing .....                                  | 2-814 |
| <i>stat()</i>                   | – get file status information using a pathname (POSIX) .....                    | 2-815 |

|                                |                                                                                    |       |
|--------------------------------|------------------------------------------------------------------------------------|-------|
| <i>statfs()</i>                | – get file status information using a pathname (POSIX) .....                       | 2-815 |
| <i>stdioFp()</i>               | – return the standard input/output/error FILE of the current task .....            | 2-816 |
| <i>stdioInit()</i>             | – initialize standard I/O support .....                                            | 2-816 |
| <i>stdioShow()</i>             | – display file pointer internals .....                                             | 2-817 |
| <i>stdioShowInit()</i>         | – initialize the standard I/O show facility .....                                  | 2-817 |
| <i>strcat()</i>                | – concatenate one string to another (ANSI) .....                                   | 2-818 |
| <i>strchr()</i>                | – find the first occurrence of a character in a string (ANSI) .....                | 2-818 |
| <i>strcmp()</i>                | – compare two strings lexicographically (ANSI) .....                               | 2-819 |
| <i>strcoll()</i>               | – compare two strings as appropriate to LC_COLLATE (ANSI) .....                    | 2-819 |
| <i>strcpy()</i>                | – copy one string to another (ANSI) .....                                          | 2-820 |
| <i>strcspn()</i>               | – return the string length up to the first character from a given set (ANSI) ..... | 2-820 |
| <i>strerror()</i>              | – map an error number to an error string (ANSI) .....                              | 2-821 |
| <i>strerror_r()</i>            | – map an error number to an error string (POSIX) .....                             | 2-821 |
| <i>strftime()</i>              | – convert broken-down time into a formatted string (ANSI) .....                    | 2-822 |
| <i>strlen()</i>                | – determine the length of a string (ANSI) .....                                    | 2-823 |
| <i>strncat()</i>               | – concatenate characters from one string to another (ANSI) .....                   | 2-824 |
| <i>strncmp()</i>               | – compare the first n characters of two strings (ANSI) .....                       | 2-824 |
| <i>strncpy()</i>               | – copy characters from one string to another (ANSI) .....                          | 2-825 |
| <i>strpbrk()</i>               | – find the first occurrence in a string of a character from a given set (ANSI) ... | 2-825 |
| <i>strrchr()</i>               | – find the last occurrence of a character in a string (ANSI) .....                 | 2-826 |
| <i>strspn()</i>                | – return the string length up to the first character not in a given set (ANSI) ... | 2-826 |
| <i>strstr()</i>                | – find the first occurrence of a substring in a string (ANSI) .....                | 2-827 |
| <i>strtod()</i>                | – convert the initial portion of a string to a double (ANSI) .....                 | 2-827 |
| <i>strtok()</i>                | – break down a string into tokens (ANSI) .....                                     | 2-828 |
| <i>strtok_r()</i>              | – break down a string into tokens (reentrant) (POSIX) .....                        | 2-829 |
| <i>strtol()</i>                | – convert a string to a long integer (ANSI) .....                                  | 2-830 |
| <i>strtoul()</i>               | – convert a string to an unsigned long integer (ANSI) .....                        | 2-831 |
| <i>strxfrm()</i>               | – transform up to n characters of s2 into s1 (ANSI) .....                          | 2-833 |
| <i>swab()</i>                  | – swap bytes .....                                                                 | 2-833 |
| <i>symAdd()</i>                | – create and add a symbol to a symbol table, including a group number .....        | 2-834 |
| <i>symEach()</i>               | – call a routine to examine each entry in a symbol table .....                     | 2-834 |
| <i>symFindByName()</i>         | – look up a symbol by name .....                                                   | 2-835 |
| <i>symFindByNameAndType()</i>  | – look up a symbol by name and type .....                                          | 2-836 |
| <i>symFindByValue()</i>        | – look up a symbol by value .....                                                  | 2-836 |
| <i>symFindByValueAndType()</i> | – look up a symbol by value and type .....                                         | 2-837 |
| <i>symLibInit()</i>            | – initialize the symbol table library .....                                        | 2-838 |
| <i>symRemove()</i>             | – remove a symbol from a symbol table .....                                        | 2-838 |
| <i>symSyncLibInit()</i>        | – initialize host/target symbol table synchronization .....                        | 2-839 |
| <i>symSyncTimeoutSet()</i>     | – set WTX timeout .....                                                            | 2-839 |
| <i>symTblCreate()</i>          | – create a symbol table .....                                                      | 2-839 |
| <i>symTblDelete()</i>          | – delete a symbol table .....                                                      | 2-840 |
| <i>sysAuxClkConnect()</i>      | – connect a routine to the auxiliary clock interrupt .....                         | 2-841 |
| <i>sysAuxClkDisable()</i>      | – turn off auxiliary clock interrupts .....                                        | 2-841 |
| <i>sysAuxClkEnable()</i>       | – turn on auxiliary clock interrupts .....                                         | 2-842 |
| <i>sysAuxClkRateGet()</i>      | – get the auxiliary clock rate .....                                               | 2-842 |
| <i>sysAuxClkRateSet()</i>      | – set the auxiliary clock rate .....                                               | 2-843 |
| <i>sysBspRev()</i>             | – return the BSP version and revision number .....                                 | 2-843 |
| <i>sysBusIntAck()</i>          | – acknowledge a bus interrupt .....                                                | 2-844 |

|                               |                                                                            |       |
|-------------------------------|----------------------------------------------------------------------------|-------|
| <i>sysBusIntGen()</i>         | – generate a bus interrupt .....                                           | 2-844 |
| <i>sysBusTas()</i>            | – test and set a location across the bus .....                             | 2-845 |
| <i>sysBusToLocalAdrs()</i>    | – convert a bus address to a local address .....                           | 2-845 |
| <i>sysClkConnect()</i>        | – connect a routine to the system clock interrupt .....                    | 2-846 |
| <i>sysClkDisable()</i>        | – turn off system clock interrupts .....                                   | 2-846 |
| <i>sysClkEnable()</i>         | – turn on system clock interrupts .....                                    | 2-847 |
| <i>sysClkRateGet()</i>        | – get the system clock rate .....                                          | 2-847 |
| <i>sysClkRateSet()</i>        | – set the system clock rate .....                                          | 2-848 |
| <i>sysHwInit()</i>            | – initialize the system hardware .....                                     | 2-848 |
| <i>sysIntDisable()</i>        | – disable a bus interrupt level .....                                      | 2-849 |
| <i>sysIntEnable()</i>         | – enable a bus interrupt level .....                                       | 2-849 |
| <i>sysLocalToBusAdrs()</i>    | – convert a local address to a bus address .....                           | 2-850 |
| <i>sysMailboxConnect()</i>    | – connect a routine to the mailbox interrupt .....                         | 2-850 |
| <i>sysMailboxEnable()</i>     | – enable the mailbox interrupt .....                                       | 2-851 |
| <i>sysMemTop()</i>            | – get the address of the top of logical memory .....                       | 2-851 |
| <i>sysModel()</i>             | – return the model name of the CPU board .....                             | 2-852 |
| <i>sysNvRamGet()</i>          | – get the contents of non-volatile RAM .....                               | 2-852 |
| <i>sysNvRamSet()</i>          | – write to non-volatile RAM .....                                          | 2-853 |
| <i>sysPhysMemTop()</i>        | – get the address of the top of memory .....                               | 2-853 |
| <i>sysProcNumGet()</i>        | – get the processor number .....                                           | 2-854 |
| <i>sysProcNumSet()</i>        | – set the processor number .....                                           | 2-854 |
| <i>sysScsiBusReset()</i>      | – assert the RST line on the SCSI bus (Western Digital WD33C93 only) ..... | 2-855 |
| <i>sysScsiConfig()</i>        | – system SCSI configuration .....                                          | 2-855 |
| <i>sysScsiInit()</i>          | – initialize an on-board SCSI port .....                                   | 2-857 |
| <i>sysSerialChanGet()</i>     | – get the SIO_CHAN device associated with a serial channel .....           | 2-857 |
| <i>sysSerialHwInit()</i>      | – initialize the BSP serial devices to a quiescent state .....             | 2-858 |
| <i>sysSerialHwInit2()</i>     | – connect BSP serial device interrupts .....                               | 2-858 |
| <i>sysSerialReset()</i>       | – reset all SIO devices to a quiet state .....                             | 2-859 |
| <i>system()</i>               | – pass a string to a command processor (Unimplemented) (ANSI) .....        | 2-859 |
| <i>sysToMonitor()</i>         | – transfer control to the ROM monitor .....                                | 2-860 |
| <i>tan()</i>                  | – compute a tangent (ANSI) .....                                           | 2-860 |
| <i>tanf()</i>                 | – compute a tangent (ANSI) .....                                           | 2-861 |
| <i>tanh()</i>                 | – compute a hyperbolic tangent (ANSI) .....                                | 2-861 |
| <i>tanhf()</i>                | – compute a hyperbolic tangent (ANSI) .....                                | 2-862 |
| <i>tapeFsDevInit()</i>        | – associate a sequential device with tape volume functions .....           | 2-862 |
| <i>tapeFsInit()</i>           | – initialize the tape volume library .....                                 | 2-863 |
| <i>tapeFsReadyChange()</i>    | – notify tapeFsLib of a change in ready status .....                       | 2-863 |
| <i>tapeFsVolUnmount()</i>     | – disable a tape device volume .....                                       | 2-864 |
| <i>taskActivate()</i>         | – activate a task that has been initialized .....                          | 2-865 |
| <i>taskCreateHookAdd()</i>    | – add a routine to be called at every task create .....                    | 2-865 |
| <i>taskCreateHookDelete()</i> | – delete a previously added task create routine .....                      | 2-866 |
| <i>taskCreateHookShow()</i>   | – show the list of task create routines .....                              | 2-866 |
| <i>taskDelay()</i>            | – delay a task from executing .....                                        | 2-866 |
| <i>taskDelete()</i>           | – delete a task .....                                                      | 2-867 |
| <i>taskDeleteForce()</i>      | – delete a task without restriction .....                                  | 2-868 |
| <i>taskDeleteHookAdd()</i>    | – add a routine to be called at every task delete .....                    | 2-868 |
| <i>taskDeleteHookDelete()</i> | – delete a previously added task delete routine .....                      | 2-869 |
| <i>taskDeleteHookShow()</i>   | – show the list of task delete routines .....                              | 2-869 |

|                               |                                                                 |       |
|-------------------------------|-----------------------------------------------------------------|-------|
| <i>taskHookInit()</i>         | – initialize task hook facilities .....                         | 2-870 |
| <i>taskHookShowInit()</i>     | – initialize the task hook show facility .....                  | 2-870 |
| <i>taskIdDefault()</i>        | – set the default task ID .....                                 | 2-871 |
| <i>taskIdListGet()</i>        | – get a list of active task IDs .....                           | 2-871 |
| <i>taskIdSelf()</i>           | – get the task ID of a running task .....                       | 2-872 |
| <i>taskIdVerify()</i>         | – verify the existence of a task .....                          | 2-872 |
| <i>taskInfoGet()</i>          | – get information about a task .....                            | 2-873 |
| <i>taskInit()</i>             | – initialize a task with a stack at a specified address .....   | 2-873 |
| <i>taskIsReady()</i>          | – check if a task is ready to run .....                         | 2-874 |
| <i>taskIsSuspended()</i>      | – check if a task is suspended .....                            | 2-875 |
| <i>taskLock()</i>             | – disable task rescheduling .....                               | 2-875 |
| <i>taskName()</i>             | – get the name associated with a task ID .....                  | 2-876 |
| <i>taskNameToId()</i>         | – look up the task ID associated with a task name .....         | 2-876 |
| <i>taskOptionsGet()</i>       | – examine task options .....                                    | 2-877 |
| <i>taskOptionsSet()</i>       | – change task options .....                                     | 2-877 |
| <i>taskPriorityGet()</i>      | – examine the priority of a task .....                          | 2-878 |
| <i>taskPrioritySet()</i>      | – change the priority of a task .....                           | 2-878 |
| <i>taskRegsGet()</i>          | – get a task’s registers from the TCB .....                     | 2-879 |
| <i>taskRegsSet()</i>          | – set a task’s registers .....                                  | 2-879 |
| <i>taskRegsShow()</i>         | – display the contents of a task’s registers .....              | 2-880 |
| <i>taskRestart()</i>          | – restart a task .....                                          | 2-881 |
| <i>taskResume()</i>           | – resume a task .....                                           | 2-881 |
| <i>taskSafe()</i>             | – make the calling task safe from deletion .....                | 2-882 |
| <i>taskShow()</i>             | – display task information from TCBs .....                      | 2-882 |
| <i>taskShowInit()</i>         | – initialize the task show routine facility .....               | 2-883 |
| <i>taskSpawn()</i>            | – spawn a task .....                                            | 2-884 |
| <i>taskSRInit()</i>           | – initialize the default task status register (MIPS) .....      | 2-885 |
| <i>taskSRSet()</i>            | – set the task status register (MC680x0, MIPS, i386/i486) ..... | 2-886 |
| <i>taskStatusString()</i>     | – get a task’s status as a string .....                         | 2-886 |
| <i>taskSuspend()</i>          | – suspend a task .....                                          | 2-887 |
| <i>taskSwitchHookAdd()</i>    | – add a routine to be called at every task switch .....         | 2-888 |
| <i>taskSwitchHookDelete()</i> | – delete a previously added task switch routine .....           | 2-889 |
| <i>taskSwitchHookShow()</i>   | – show the list of task switch routines .....                   | 2-889 |
| <i>taskTcb()</i>              | – get the task control block for a task ID .....                | 2-889 |
| <i>taskUnlock()</i>           | – enable task rescheduling .....                                | 2-890 |
| <i>taskUnsafe()</i>           | – make the calling task unsafe from deletion .....              | 2-890 |
| <i>taskVarAdd()</i>           | – add a task variable to a task .....                           | 2-891 |
| <i>taskVarDelete()</i>        | – remove a task variable from a task .....                      | 2-892 |
| <i>taskVarGet()</i>           | – get the value of a task variable .....                        | 2-893 |
| <i>taskVarInfo()</i>          | – get a list of task variables of a task .....                  | 2-893 |
| <i>taskVarInit()</i>          | – initialize the task variables facility .....                  | 2-894 |
| <i>taskVarSet()</i>           | – set the value of a task variable .....                        | 2-895 |
| <i>tcicInit()</i>             | – initialize the TCIC chip .....                                | 2-895 |
| <i>tcicShow()</i>             | – show all configurations of the TCIC chip .....                | 2-896 |
| <i>tcpDebugShow()</i>         | – display debugging information for the TCP protocol .....      | 2-896 |
| <i>tcpShowInit()</i>          | – initialize TCP show routines .....                            | 2-897 |
| <i>tcpstatShow()</i>          | – display all statistics for the TCP protocol .....             | 2-897 |
| <i>tcw()</i>                  | – return the contents of the tcw register (i960) .....          | 2-897 |

|                               |                                                                               |       |
|-------------------------------|-------------------------------------------------------------------------------|-------|
| <i>td()</i>                   | – delete a task .....                                                         | 2-898 |
| <i>telnetd()</i>              | – VxWorks telnet daemon .....                                                 | 2-898 |
| <i>telnetInit()</i>           | – initialize the telnet daemon .....                                          | 2-899 |
| <i>tftpCopy()</i>             | – transfer a file via TFTP .....                                              | 2-899 |
| <i>tftpdDirectoryAdd()</i>    | – add a directory to the access list .....                                    | 2-900 |
| <i>tftpdDirectoryRemove()</i> | – delete a directory from the access list .....                               | 2-900 |
| <i>tftpdInit()</i>            | – initialize the TFTP server task .....                                       | 2-901 |
| <i>tftpdTask()</i>            | – TFTP server daemon task .....                                               | 2-901 |
| <i>tftpGet()</i>              | – get a file from a remote system .....                                       | 2-902 |
| <i>tftpInfoShow()</i>         | – get TFTP status information .....                                           | 2-903 |
| <i>tftpInit()</i>             | – initialize a TFTP session .....                                             | 2-903 |
| <i>tftpModeSet()</i>          | – set the TFTP transfer mode .....                                            | 2-904 |
| <i>tftpPeerSet()</i>          | – set the TFTP server address .....                                           | 2-904 |
| <i>tftpPut()</i>              | – put a file to a remote system .....                                         | 2-905 |
| <i>tftpQuit()</i>             | – quit a TFTP session .....                                                   | 2-905 |
| <i>tftpSend()</i>             | – send a TFTP message to the remote system .....                              | 2-906 |
| <i>tftpXfer()</i>             | – transfer a file via TFTP using a stream interface .....                     | 2-907 |
| <i>ti()</i>                   | – print complete information from a task's TCB .....                          | 2-908 |
| <i>tickAnnounce()</i>         | – announce a clock tick to the kernel .....                                   | 2-909 |
| <i>tickGet()</i>              | – get the value of the kernel's tick counter .....                            | 2-909 |
| <i>tickSet()</i>              | – set the value of the kernel's tick counter .....                            | 2-910 |
| <i>time()</i>                 | – determine the current calendar time (ANSI) .....                            | 2-910 |
| <i>timer_cancel()</i>         | – cancel a timer .....                                                        | 2-911 |
| <i>timer_connect()</i>        | – connect a user routine to the timer signal .....                            | 2-911 |
| <i>timer_create()</i>         | – allocate a timer using the specified clock for a timing base (POSIX) .....  | 2-912 |
| <i>timer_delete()</i>         | – remove a previously created timer (POSIX) .....                             | 2-913 |
| <i>timer_getoverrun()</i>     | – return the timer expiration overrun (POSIX) .....                           | 2-913 |
| <i>timer_gettime()</i>        | – get the remaining time before expiration and the reload value (POSIX) ..... | 2-914 |
| <i>timer_settime()</i>        | – set the time until the next expiration and arm timer (POSIX) .....          | 2-914 |
| <i>timex()</i>                | – time a single execution of a function or functions .....                    | 2-915 |
| <i>timexClear()</i>           | – clear the list of function calls to be timed .....                          | 2-916 |
| <i>timexFunc()</i>            | – specify functions to be timed .....                                         | 2-916 |
| <i>timexHelp()</i>            | – display synopsis of execution timer facilities .....                        | 2-917 |
| <i>timexInit()</i>            | – include the execution timer library .....                                   | 2-918 |
| <i>timexN()</i>               | – time repeated executions of a function or group of functions .....          | 2-918 |
| <i>timexPost()</i>            | – specify functions to be called after timing .....                           | 2-919 |
| <i>timexPre()</i>             | – specify functions to be called prior to timing .....                        | 2-919 |
| <i>timexShow()</i>            | – display the list of function calls to be timed .....                        | 2-920 |
| <i>tmpfile()</i>              | – create a temporary binary file (Unimplemented) (ANSI) .....                 | 2-920 |
| <i>tmpnam()</i>               | – generate a temporary file name (ANSI) .....                                 | 2-921 |
| <i>tolower()</i>              | – convert an upper-case letter to its lower-case equivalent (ANSI) .....      | 2-921 |
| <i>toupper()</i>              | – convert a lower-case letter to its upper-case equivalent (ANSI) .....       | 2-922 |
| <i>tr()</i>                   | – resume a task .....                                                         | 2-922 |
| <i>trunc()</i>                | – truncate to integer .....                                                   | 2-923 |
| <i>truncf()</i>               | – truncate to integer .....                                                   | 2-923 |
| <i>ts()</i>                   | – suspend a task .....                                                        | 2-924 |
| <i>tsp()</i>                  | – return the contents of register sp (i960) .....                             | 2-924 |
| <i>tt()</i>                   | – display a stack trace of a task .....                                       | 2-925 |

|                             |                                                                         |       |
|-----------------------------|-------------------------------------------------------------------------|-------|
| <i>ttyDevCreate()</i>       | – create a VxWorks device for a serial channel .....                    | 2-926 |
| <i>ttyDrv()</i>             | – initialize the tty driver .....                                       | 2-926 |
| <i>tyAbortFuncSet()</i>     | – set the abort function .....                                          | 2-927 |
| <i>tyAbortSet()</i>         | – change the abort character .....                                      | 2-927 |
| <i>tyBackspaceSet()</i>     | – change the backspace character .....                                  | 2-928 |
| <i>tyDeleteLineSet()</i>    | – change the line-delete character .....                                | 2-928 |
| <i>tyDevInit()</i>          | – initialize the tty device descriptor .....                            | 2-929 |
| <i>tyEOFSet()</i>           | – change the end-of-file character .....                                | 2-929 |
| <i>tyIoctl()</i>            | – handle device control requests .....                                  | 2-930 |
| <i>tyIRd()</i>              | – interrupt-level input .....                                           | 2-930 |
| <i>tyITx()</i>              | – interrupt-level output .....                                          | 2-931 |
| <i>tyMonitorTrapSet()</i>   | – change the trap-to-monitor character .....                            | 2-931 |
| <i>tyRead()</i>             | – do a task-level read for a tty device .....                           | 2-932 |
| <i>tyWrite()</i>            | – do a task-level write for a tty device .....                          | 2-933 |
| <i>udpShowInit()</i>        | – initialize UDP show routines .....                                    | 2-933 |
| <i>udpstatShow()</i>        | – display statistics for the UDP protocol .....                         | 2-933 |
| <i>ulattach()</i>           | – attach a ULIP interface to a list of network interfaces (VxSim) ..... | 2-934 |
| <i>ulipDebugSet()</i>       | – Set debug flag in UNIX's ULIP driver .....                            | 2-934 |
| <i>ulipDelete()</i>         | – delete a ULIP interface (VxSim) .....                                 | 2-935 |
| <i>ulipInit()</i>           | – initialize the ULIP interface (VxSim) .....                           | 2-935 |
| <i>ulStartOutput()</i>      | – push packets onto "interface" .....                                   | 2-936 |
| <i>ultraAddrFilterSet()</i> | – set the address filter for multicast addresses .....                  | 2-936 |
| <i>ultraattach()</i>        | – publish ultra interface and initialize device .....                   | 2-937 |
| <i>ultraLoad()</i>          | – initialize the driver and device .....                                | 2-937 |
| <i>ultraMemInit()</i>       | – initialize memory for the chip .....                                  | 2-938 |
| <i>ultraParse()</i>         | – parse the init string .....                                           | 2-938 |
| <i>ultraPut()</i>           | – copy a packet to the interface. ....                                  | 2-939 |
| <i>ultraShow()</i>          | – display statistics for the ultra network interface .....              | 2-940 |
| <i>ungetc()</i>             | – push a character back into an input stream (ANSI) .....               | 2-940 |
| <i>unixDevInit()</i>        | – initialize a UNIX_DUSART .....                                        | 2-941 |
| <i>unixDevInit2()</i>       | – enable interrupts .....                                               | 2-942 |
| <i>unixDiskDevCreate()</i>  | – create a UNIX disk device .....                                       | 2-942 |
| <i>unixDiskInit()</i>       | – initialize a dosFs disk on top of UNIX .....                          | 2-943 |
| <i>unixDrv()</i>            | – install UNIX disk driver .....                                        | 2-943 |
| <i>unixIntRcv()</i>         | – handle a channel's receive-character interrupt. ....                  | 2-944 |
| <i>unld()</i>               | – unload an object module by specifying a file name or module ID .....  | 2-944 |
| <i>unldByGroup()</i>        | – unload an object module by specifying a group number .....            | 2-945 |
| <i>unldByModuleId()</i>     | – unload an object module by specifying a module ID .....               | 2-945 |
| <i>unldByNameAndPath()</i>  | – unload an object module by specifying a name and path .....           | 2-946 |
| <i>unlink()</i>             | – delete a file (POSIX) .....                                           | 2-946 |
| <i>usrAtaConfig()</i>       | – mount a DOS file system from an ATA hard disk .....                   | 2-947 |
| <i>usrAtaPartition()</i>    | – get an offset to the first partition of the drive .....               | 2-947 |
| <i>usrClock()</i>           | – user-defined system clock interrupt routine .....                     | 2-948 |
| <i>usrFdConfig()</i>        | – mount a DOS file system from a floppy disk .....                      | 2-948 |
| <i>usrIdeConfig()</i>       | – mount a DOS file system from an IDE hard disk .....                   | 2-949 |
| <i>usrInit()</i>            | – user-defined system initialization routine .....                      | 2-950 |
| <i>usrRoot()</i>            | – the root task .....                                                   | 2-950 |
| <i>usrScsiConfig()</i>      | – configure SCSI peripherals .....                                      | 2-951 |

|                              |                                                                                           |       |
|------------------------------|-------------------------------------------------------------------------------------------|-------|
| <i>usrSmObjInit()</i>        | – initialize shared memory objects .....                                                  | 2-951 |
| <i>uswab()</i>               | – swap bytes with buffers that are not necessarily aligned .....                          | 2-952 |
| <i>utime()</i>               | – update time on a file .....                                                             | 2-953 |
| <i>va_arg()</i>              | – expand to an expression having the type and value of the call's next argument .....     | 2-953 |
| <i>va_end()</i>              | – facilitate a normal return from a routine using a <i>va_list</i> object .....           | 2-954 |
| <i>va_start()</i>            | – initialize a <i>va_list</i> object for use by <i>va_arg()</i> and <i>va_end()</i> ..... | 2-954 |
| <i>valloc()</i>              | – allocate memory on a page boundary .....                                                | 2-955 |
| <i>version()</i>             | – print VxWorks version information .....                                                 | 2-955 |
| <i>vsprintf()</i>            | – write a string formatted with a variable argument list to a file descriptor .....       | 2-956 |
| <i>vfprintf()</i>            | – write a formatted string to a stream (ANSI) .....                                       | 2-956 |
| <i>vmBaseGlobalMapInit()</i> | – initialize global mapping .....                                                         | 2-957 |
| <i>vmBaseLibInit()</i>       | – initialize base virtual memory support .....                                            | 2-958 |
| <i>vmBasePageSizeGet()</i>   | – return the page size .....                                                              | 2-958 |
| <i>vmBaseStateSet()</i>      | – change the state of a block of virtual memory .....                                     | 2-959 |
| <i>vmContextCreate()</i>     | – create a new virtual memory context (VxVMI Opt.) .....                                  | 2-960 |
| <i>vmContextDelete()</i>     | – delete a virtual memory context (VxVMI Opt.) .....                                      | 2-960 |
| <i>vmContextShow()</i>       | – display the translation table for a context (VxVMI Opt.) .....                          | 2-961 |
| <i>vmCurrentGet()</i>        | – get the current virtual memory context (VxVMI Opt.) .....                               | 2-961 |
| <i>vmCurrentSet()</i>        | – set the current virtual memory context (VxVMI Opt.) .....                               | 2-962 |
| <i>vmEnable()</i>            | – enable or disable virtual memory (VxVMI Opt.) .....                                     | 2-962 |
| <i>vmGlobalInfoGet()</i>     | – get global virtual memory information (VxVMI Opt.) .....                                | 2-963 |
| <i>vmGlobalMap()</i>         | – map physical pages to virtual space in shared global virtual mem (VxVMI Opt.) .....     | 2-963 |
| <i>vmGlobalMapInit()</i>     | – initialize global mapping (VxVMI Opt.) .....                                            | 2-964 |
| <i>vmLibInit()</i>           | – initialize the virtual memory support module (VxVMI Opt.) .....                         | 2-965 |
| <i>vmMap()</i>               | – map physical space into virtual space (VxVMI Opt.) .....                                | 2-966 |
| <i>vmPageBlockSizeGet()</i>  | – get the architecture-dependent page block size (VxVMI Opt.) .....                       | 2-967 |
| <i>vmPageSizeGet()</i>       | – return the page size (VxVMI Opt.) .....                                                 | 2-967 |
| <i>vmShowInit()</i>          | – include virtual memory show facility (VxVMI Opt.) .....                                 | 2-968 |
| <i>vmStateGet()</i>          | – get the state of a page of virtual memory (VxVMI Opt.) .....                            | 2-968 |
| <i>vmStateSet()</i>          | – change the state of a block of virtual memory (VxVMI Opt.) .....                        | 2-969 |
| <i>vmTextProtect()</i>       | – write-protect a text segment (VxVMI Opt.) .....                                         | 2-970 |
| <i>vmTranslate()</i>         | – translate a virtual address to a physical address (VxVMI Opt.) .....                    | 2-971 |
| <i>vprintf()</i>             | – write string formatted with variable argument list to standard output (ANSI) .....      | 2-971 |
| <i>vsprintf()</i>            | – write a string formatted with a variable argument list to a buffer (ANSI) .....         | 2-972 |
| <i>vxMemArchProbe()</i>      | – architecture specific part of vxMemProbe .....                                          | 2-972 |
| <i>vxMemProbe()</i>          | – probe an address for a bus error .....                                                  | 2-973 |
| <i>vxMemProbeAsi()</i>       | – probe address in ASI space for bus error (SPARC) .....                                  | 2-974 |
| <i>vxPowerDown()</i>         | – place the processor in reduced-power mode (PowerPC) .....                               | 2-975 |
| <i>vxPowerModeGet()</i>      | – get the power management mode (PowerPC) .....                                           | 2-975 |
| <i>vxPowerModeSet()</i>      | – set the power management mode (PowerPC) .....                                           | 2-976 |
| <i>vxSSDisable()</i>         | – disable the superscalar dispatch (MC68060) .....                                        | 2-977 |
| <i>vxSSEnable()</i>          | – enable the superscalar dispatch (MC68060) .....                                         | 2-977 |
| <i>vxTas()</i>               | – C-callable atomic test-and-set primitive .....                                          | 2-977 |
| <i>VXWBSem::VXWBSem()</i>    | – create and initialize a binary semaphore .....                                          | 2-978 |
| <i>VXWCSem::VXWCSem()</i>    | – create and initialize a counting semaphore .....                                        | 2-980 |
| <i>VXWList::add()</i>        | – add a node to the end of list .....                                                     | 2-981 |
| <i>VXWList::concat()</i>     | – concatenate two lists .....                                                             | 2-981 |
| <i>VXWList::count()</i>      | – report the number of nodes in a list (WFC Opt.) .....                                   | 2-982 |



|                                   |                                                                            |        |
|-----------------------------------|----------------------------------------------------------------------------|--------|
| <i>VXWList::extract()</i>         | – extract a sublist from list (WFC Opt.) .....                             | 2-982  |
| <i>VXWList::find()</i>            | – find a node in list (WFC Opt.) .....                                     | 2-982  |
| <i>VXWList::first()</i>           | – find first node in list (WFC Opt.) .....                                 | 2-983  |
| <i>VXWList::get()</i>             | – delete and return the first node from list (WFC Opt.) .....              | 2-983  |
| <i>VXWList::insert()</i>          | – insert a node in list after a specified node (WFC Opt.) .....            | 2-984  |
| <i>VXWList::last()</i>            | – find the last node in list (WFC Opt.) .....                              | 2-984  |
| <i>VXWList::next()</i>            | – find the next node in list (WFC Opt.) .....                              | 2-984  |
| <i>VXWList::nStep()</i>           | – find a list node nStep steps away from a specified node (WFC Opt.) ..... | 2-985  |
| <i>VXWList::nth()</i>             | – find the Nth node in a list (WFC Opt.) .....                             | 2-985  |
| <i>VXWList::previous()</i>        | – find the previous node in list (WFC Opt.) .....                          | 2-986  |
| <i>VXWList::remove()</i>          | – delete a specified node from list (WFC Opt.) .....                       | 2-986  |
| <i>VXWList::VXWList()</i>         | – initialize a list (WFC Opt.) .....                                       | 2-986  |
| <i>VXWList::VXWList()</i>         | – initialize a list as a copy of another (WFC Opt.) .....                  | 2-987  |
| <i>VXWList::~VXWList()</i>        | – free up a list (WFC Opt.) .....                                          | 2-987  |
| <i>VXWMemPart::addToPool()</i>    | – add memory to a memory partition (WFC Opt.) .....                        | 2-988  |
| <i>VXWMemPart::alignedAlloc()</i> | – allocate aligned memory from partition (WFC Opt.) .....                  | 2-988  |
| <i>VXWMemPart::alloc()</i>        | – allocate a block of memory from partition (WFC Opt.) .....               | 2-989  |
| <i>VXWMemPart::findMax()</i>      | – find the size of the largest available free block (WFC Opt.) .....       | 2-989  |
| <i>VXWMemPart::free()</i>         | – free a block of memory in partition (WFC Opt.) .....                     | 2-989  |
| <i>VXWMemPart::info()</i>         | – get partition information (WFC Opt.) .....                               | 2-990  |
| <i>VXWMemPart::options()</i>      | – set the debug options for memory partition (WFC Opt.) .....              | 2-990  |
| <i>VXWMemPart::realloc()</i>      | – reallocate a block of memory in partition (WFC Opt.) .....               | 2-991  |
| <i>VXWMemPart::show()</i>         | – show partition blocks and statistics (WFC Opt.) .....                    | 2-992  |
| <i>VXWMemPart::VXWMemPart()</i>   | – create a memory partition (WFC Opt.) .....                               | 2-992  |
| <i>VXWModule::flags()</i>         | – get the flags associated with this module (WFC Opt.) .....               | 2-993  |
| <i>VXWModule::info()</i>          | – get information about object module (WFC Opt.) .....                     | 2-993  |
| <i>VXWModule::name()</i>          | – get the name associated with module (WFC Opt.) .....                     | 2-993  |
| <i>VXWModule::segFirst()</i>      | – find the first segment in module (WFC Opt.) .....                        | 2-994  |
| <i>VXWModule::segGet()</i>        | – get (delete and return) the first segment from module (WFC Opt.) .....   | 2-994  |
| <i>VXWModule::segNext()</i>       | – find the next segment in module (WFC Opt.) .....                         | 2-994  |
| <i>VXWModule::VXWModule()</i>     | – build module object from module ID (WFC Opt.) .....                      | 2-995  |
| <i>VXWModule::VXWModule()</i>     | – load an object module at specified memory addresses (WFC Opt.) .....     | 2-995  |
| <i>VXWModule::VXWModule()</i>     | – load an object module into memory (WFC Opt.) .....                       | 2-997  |
| <i>VXWModule::VXWModule()</i>     | – create and initialize an object module (WFC Opt.) .....                  | 2-998  |
| <i>VXWModule::~VXWModule()</i>    | – unload an object module (WFC Opt.) .....                                 | 2-998  |
| <i>VXWMSem::giveForce()</i>       | – give a mutual-exclusion semaphore without restrictions (WFC Opt.) .....  | 2-999  |
| <i>VXWMSem::VXWMSem()</i>         | – create and initialize a mutual-exclusion semaphore (WFC Opt.) .....      | 2-999  |
| <i>VXWMsgQ::info()</i>            | – get information about message queue (WFC Opt.) .....                     | 2-1002 |
| <i>VXWMsgQ::numMsgs()</i>         | – report the number of messages queued (WFC Opt.) .....                    | 2-1004 |
| <i>VXWMsgQ::receive()</i>         | – receive a message from message queue (WFC Opt.) .....                    | 2-1004 |
| <i>VXWMsgQ::send()</i>            | – send a message to message queue (WFC Opt.) .....                         | 2-1005 |
| <i>VXWMsgQ::show()</i>            | – show information about a message queue (WFC Opt.) .....                  | 2-1007 |
| <i>VXWMsgQ::VXWMsgQ()</i>         | – create and initialize a message queue (WFC Opt.) .....                   | 2-1008 |
| <i>VXWMsgQ::VXWMsgQ()</i>         | – build message-queue object from ID (WFC Opt.) .....                      | 2-1008 |
| <i>VXWMsgQ::~VXWMsgQ()</i>        | – delete message queue (WFC Opt.) .....                                    | 2-1009 |
| <i>VXWRingBuf::flush()</i>        | – make ring buffer empty (WFC Opt.) .....                                  | 2-1009 |
| <i>VXWRingBuf::freeBytes()</i>    | – determine the number of free bytes in ring buffer (WFC Opt.) .....       | 2-1010 |
| <i>VXWRingBuf::get()</i>          | – get characters from ring buffer (WFC Opt.) .....                         | 2-1010 |

|                                              |                                                                           |        |
|----------------------------------------------|---------------------------------------------------------------------------|--------|
| <code>VXWRingBuf::isEmpty()</code>           | – test whether ring buffer is empty (WFC Opt.)                            | 2-1010 |
| <code>VXWRingBuf::isFull()</code>            | – test whether ring buffer is full (no more room) (WFC Opt.)              | 2-1011 |
| <code>VXWRingBuf::moveAhead()</code>         | – advance ring pointer by n bytes (WFC Opt.)                              | 2-1011 |
| <code>VXWRingBuf::nBytes()</code>            | – determine the number of bytes in ring buffer (WFC Opt.)                 | 2-1012 |
| <code>VXWRingBuf::put()</code>               | – put bytes into ring buffer (WFC Opt.)                                   | 2-1012 |
| <code>VXWRingBuf::putAhead()</code>          | – put a byte ahead in ring buffer without moving ring pointers (WFC Opt.) | 2-1013 |
| <code>VXWRingBuf::VXWRingBuf()</code>        | – create an empty ring buffer (WFC Opt.)                                  | 2-1013 |
| <code>VXWRingBuf::VXWRingBuf()</code>        | – build ring-buffer object from existing ID (WFC Opt.)                    | 2-1014 |
| <code>VXWRingBuf::~VXWRingBuf()</code>       | – delete ring buffer (WFC Opt.)                                           | 2-1014 |
| <code>VXWSem::flush()</code>                 | – unblock every task pending on a semaphore (WFC Opt.)                    | 2-1014 |
| <code>VXWSem::give()</code>                  | – give a semaphore (WFC Opt.)                                             | 2-1015 |
| <code>VXWSem::id()</code>                    | – reveal underlying semaphore ID (WFC Opt.)                               | 2-1015 |
| <code>VXWSem::info()</code>                  | – get a list of task IDs that are blocked on a semaphore (WFC Opt.)       | 2-1016 |
| <code>VXWSem::show()</code>                  | – show information about a semaphore (WFC Opt.)                           | 2-1016 |
| <code>VXWSem::take()</code>                  | – take a semaphore (WFC Opt.)                                             | 2-1017 |
| <code>VXWSem::VXWSem()</code>                | – build semaphore object from semaphore ID (WFC Opt.)                     | 2-1018 |
| <code>VXWSem::~VXWSem()</code>               | – delete a semaphore (WFC Opt.)                                           | 2-1018 |
| <code>VXWSmName::nameGet()</code>            | – get name and type of a shared memory object (VxMP, WFC Opt.)            | 2-1019 |
| <code>VXWSmName::nameGet()</code>            | – get name of a shared memory object (VxMP, WFC Opt.)                     | 2-1019 |
| <code>VXWSmName::nameSet()</code>            | – define a name in the shared-memory name database (VxMP, WFC Opt.)       | 2-1020 |
| <code>VXWSmName::~VXWSmName()</code>         | – remove object from shared memory name database (VxMP, WFC Opt.)         | 2-1021 |
| <code>VXWSymTab::add()</code>                | – create and add symbol to symbol table, including group no. (WFC Opt.)   | 2-1021 |
| <code>VXWSymTab::each()</code>               | – call a routine to examine each entry in a symbol table (WFC Opt.)       | 2-1022 |
| <code>VXWSymTab::findByName()</code>         | – look up a symbol by name (WFC Opt.)                                     | 2-1023 |
| <code>VXWSymTab::findByNameAndType()</code>  | – look up a symbol by name and type (WFC Opt.)                            | 2-1023 |
| <code>VXWSymTab::findByValue()</code>        | – look up a symbol by value (WFC Opt.)                                    | 2-1024 |
| <code>VXWSymTab::findByValueAndType()</code> | – look up a symbol by value and type (WFC Opt.)                           | 2-1024 |
| <code>VXWSymTab::remove()</code>             | – remove a symbol from a symbol table (WFC Opt.)                          | 2-1025 |
| <code>VXWSymTab::VXWSymTab()</code>          | – create a symbol table (WFC Opt.)                                        | 2-1025 |
| <code>VXWSymTab::VXWSymTab()</code>          | – create a symbol-table object (WFC Opt.)                                 | 2-1026 |
| <code>VXWSymTab::~VXWSymTab()</code>         | – delete a symbol table (WFC Opt.)                                        | 2-1026 |
| <code>VXWTask::activate()</code>             | – activate a task (WFC Opt.)                                              | 2-1027 |
| <code>VXWTask::deleteForce()</code>          | – delete a task without restriction (WFC Opt.)                            | 2-1027 |
| <code>VXWTask::envCreate()</code>            | – create a private environment (WFC Opt.)                                 | 2-1028 |
| <code>VXWTask::errNo()</code>                | – retrieve error status value (WFC Opt.)                                  | 2-1028 |
| <code>VXWTask::errNo()</code>                | – set error status value (WFC Opt.)                                       | 2-1028 |
| <code>VXWTask::id()</code>                   | – reveal task ID (WFC Opt.)                                               | 2-1029 |
| <code>VXWTask::info()</code>                 | – get information about a task (WFC Opt.)                                 | 2-1029 |
| <code>VXWTask::isReady()</code>              | – check if task is ready to run (WFC Opt.)                                | 2-1030 |
| <code>VXWTask::isSuspended()</code>          | – check if task is suspended (WFC Opt.)                                   | 2-1030 |
| <code>VXWTask::kill()</code>                 | – send a signal to task (WFC Opt.)                                        | 2-1030 |
| <code>VXWTask::name()</code>                 | – get the name associated with a task ID (WFC Opt.)                       | 2-1031 |
| <code>VXWTask::options()</code>              | – examine task options (WFC Opt.)                                         | 2-1031 |
| <code>VXWTask::options()</code>              | – change task options (WFC Opt.)                                          | 2-1032 |
| <code>VXWTask::priority()</code>             | – examine the priority of task (WFC Opt.)                                 | 2-1032 |
| <code>VXWTask::priority()</code>             | – change the priority of a task (WFC Opt.)                                | 2-1033 |
| <code>VXWTask::registers()</code>            | – set a task's registers (WFC Opt.)                                       | 2-1033 |
| <code>VXWTask::registers()</code>            | – get task registers from the TCB (WFC Opt.)                              | 2-1034 |

|                                 |                                                                              |        |
|---------------------------------|------------------------------------------------------------------------------|--------|
| <i>VXWTask::restart()</i>       | – restart task (WFC Opt.) .....                                              | 2-1034 |
| <i>VXWTask::resume()</i>        | – resume task (WFC Opt.) .....                                               | 2-1035 |
| <i>VXWTask::show()</i>          | – display the contents of task registers (WFC Opt.) .....                    | 2-1035 |
| <i>VXWTask::show()</i>          | – display task information from TCBs (WFC Opt.) .....                        | 2-1036 |
| <i>VXWTask::sigqueue()</i>      | – send a queued signal to task (WFC Opt.) .....                              | 2-1037 |
| <i>VXWTask::SRSet()</i>         | – set the task status register (MC680x0, MIPS, i386/i486) (WFC Opt.) .....   | 2-1037 |
| <i>VXWTask::statusString()</i>  | – get task status as a string (WFC Opt.) .....                               | 2-1038 |
| <i>VXWTask::suspend()</i>       | – suspend task (WFC Opt.) .....                                              | 2-1039 |
| <i>VXWTask::tcb()</i>           | – get the task control block (WFC Opt.) .....                                | 2-1039 |
| <i>VXWTask::varAdd()</i>        | – add a task variable to task (WFC Opt.) .....                               | 2-1040 |
| <i>VXWTask::varDelete()</i>     | – remove a task variable from task (WFC Opt.) .....                          | 2-1041 |
| <i>VXWTask::varGet()</i>        | – get the value of a task variable (WFC Opt.) .....                          | 2-1041 |
| <i>VXWTask::varInfo()</i>       | – get a list of task variables (WFC Opt.) .....                              | 2-1042 |
| <i>VXWTask::varSet()</i>        | – set the value of a task variable (WFC Opt.) .....                          | 2-1042 |
| <i>VXWTask::VXWTask()</i>       | – initialize a task object (WFC Opt.) .....                                  | 2-1043 |
| <i>VXWTask::VXWTask()</i>       | – create and spawn a task (WFC Opt.) .....                                   | 2-1043 |
| <i>VXWTask::VXWTask()</i>       | – initialize a task with a specified stack (WFC Opt.) .....                  | 2-1045 |
| <i>VXWTask::~~VXWTask()</i>     | – delete a task (WFC Opt.) .....                                             | 2-1046 |
| <i>VXWwd::cancel()</i>          | – cancel a currently counting watchdog (WFC Opt.) .....                      | 2-1046 |
| <i>VXWwd::start()</i>           | – start a watchdog timer (WFC Opt.) .....                                    | 2-1047 |
| <i>VXWwd::VXWwd()</i>           | – construct a watchdog timer (WFC Opt.) .....                                | 2-1047 |
| <i>VXWwd::VXWwd()</i>           | – construct a watchdog timer (WFC Opt.) .....                                | 2-1048 |
| <i>VXWwd::~~VXWwd()</i>         | – destroy a watchdog timer (WFC Opt.) .....                                  | 2-1048 |
| <i>wcstombs()</i>               | – convert a series of wide char’s to multibyte char’s (Unimplemented) (ANSI) | 2-1049 |
| <i>wctomb()</i>                 | – convert a wide character to a multibyte character (Unimplemented) (ANSI)   | 2-1049 |
| <i>wd33c93CtrlCreate()</i>      | – create and partially initialize a WD33C93 SBIC structure .....             | 2-1050 |
| <i>wd33c93CtrlCreateScsi2()</i> | – create and partially initialize an SBIC structure .....                    | 2-1051 |
| <i>wd33c93CtrlInit()</i>        | – initialize the user-specified fields in an SBIC structure .....            | 2-1053 |
| <i>wd33c93Show()</i>            | – display the values of all readable WD33C93 chip registers .....            | 2-1054 |
| <i>wdbNetromPktDevInit()</i>    | – initialize a NETROM packet device for the WDB agent .....                  | 2-1055 |
| <i>wdbPipePktDevInit()</i>      | – initialize a pipe packet device. ....                                      | 2-1056 |
| <i>wdbSlipPktDevInit()</i>      | – initialize a SLIP packet device for a WDB agent .....                      | 2-1056 |
| <i>wdbSystemSuspend()</i>       | – suspend the system. ....                                                   | 2-1056 |
| <i>wdbTsfsDrv()</i>             | – initialize the TSFS device driver for a WDB agent .....                    | 2-1058 |
| <i>wdbUlipPktDevInit()</i>      | – initialize the WDB agent’s communication functions for ULIP .....          | 2-1058 |
| <i>wdbUserEvtLibInit()</i>      | – include the WDB user event library .....                                   | 2-1059 |
| <i>wdbUserEvtPost()</i>         | – post a user event string to host tools. ....                               | 2-1059 |
| <i>wdbVioDrv()</i>              | – initialize the tty driver for a WDB agent .....                            | 2-1060 |
| <i>wdCancel()</i>               | – cancel a currently counting watchdog .....                                 | 2-1061 |
| <i>wdCreate()</i>               | – create a watchdog timer .....                                              | 2-1061 |
| <i>wdDelete()</i>               | – delete a watchdog timer .....                                              | 2-1061 |
| <i>wdShow()</i>                 | – show information about a watchdog .....                                    | 2-1062 |
| <i>wdShowInit()</i>             | – initialize the watchdog show facility .....                                | 2-1062 |
| <i>wdStart()</i>                | – start a watchdog timer .....                                               | 2-1063 |
| <i>whoami()</i>                 | – display the current remote identity .....                                  | 2-1064 |
| <i>wim()</i>                    | – return the contents of the window invalid mask register (SPARC) .....      | 2-1064 |
| <i>winDevInit()</i>             | – initialize a WIN_CHAN .....                                                | 2-1064 |
| <i>winDevInit2()</i>            | – initialize a WIN_CHAN, part 2 .....                                        | 2-1065 |

|                            |                                                                       |        |
|----------------------------|-----------------------------------------------------------------------|--------|
| <i>winIntRcv()</i>         | – handle a channel’s receive-character interrupt .....                | 2-1065 |
| <i>winIntTx()</i>          | – transmit a single character. ....                                   | 2-1066 |
| <i>write()</i>             | – write bytes to a file .....                                         | 2-1066 |
| <i>y()</i>                 | – return the contents of the y register (SPARC) .....                 | 2-1067 |
| <i>z8530DevInit()</i>      | – initialize a Z8530_DUSART .....                                     | 2-1067 |
| <i>z8530Int()</i>          | – handle all interrupts in one vector .....                           | 2-1068 |
| <i>z8530IntEx()</i>        | – handle error interrupts .....                                       | 2-1068 |
| <i>z8530IntRd()</i>        | – handle a receiver interrupt .....                                   | 2-1069 |
| <i>z8530IntWr()</i>        | – handle a transmitter interrupt .....                                | 2-1069 |
| <i>zbufCreate()</i>        | – create an empty zbuf .....                                          | 2-1069 |
| <i>zbufCut()</i>           | – delete bytes from a zbuf .....                                      | 2-1070 |
| <i>zbufDelete()</i>        | – delete a zbuf .....                                                 | 2-1071 |
| <i>zbufDup()</i>           | – duplicate a zbuf .....                                              | 2-1071 |
| <i>zbufExtractCopy()</i>   | – copy data from a zbuf to a buffer .....                             | 2-1072 |
| <i>zbufInsert()</i>        | – insert a zbuf into another zbuf .....                               | 2-1073 |
| <i>zbufInsertBuf()</i>     | – create a zbuf segment from a buffer and insert into a zbuf .....    | 2-1073 |
| <i>zbufInsertCopy()</i>    | – copy buffer data into a zbuf .....                                  | 2-1074 |
| <i>zbufLength()</i>        | – determine the length in bytes of a zbuf .....                       | 2-1075 |
| <i>zbufSegData()</i>       | – determine the location of data in a zbuf segment .....              | 2-1075 |
| <i>zbufSegFind()</i>       | – find the zbuf segment containing a specified byte location .....    | 2-1076 |
| <i>zbufSegLength()</i>     | – determine the length of a zbuf segment .....                        | 2-1076 |
| <i>zbufSegNext()</i>       | – get the next segment in a zbuf .....                                | 2-1077 |
| <i>zbufSegPrev()</i>       | – get the previous segment in a zbuf .....                            | 2-1077 |
| <i>zbufSockBufSend()</i>   | – create a zbuf from user data and send it to a TCP socket .....      | 2-1078 |
| <i>zbufSockBufSendto()</i> | – create a zbuf from a user message and send it to a UDP socket ..... | 2-1079 |
| <i>zbufSockLibInit()</i>   | – initialize the zbuf socket interface library .....                  | 2-1080 |
| <i>zbufSockRecv()</i>      | – receive data in a zbuf from a TCP socket .....                      | 2-1080 |
| <i>zbufSockRecvfrom()</i>  | – receive a message in a zbuf from a UDP socket .....                 | 2-1081 |
| <i>zbufSockSend()</i>      | – send zbuf data to a TCP socket .....                                | 2-1082 |
| <i>zbufSockSendto()</i>    | – send a zbuf message to a UDP socket .....                           | 2-1083 |
| <i>zbufSplit()</i>         | – split a zbuf into two separate zbufs .....                          | 2-1084 |

---

## ***a0()***

|                    |                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b><i>a0()</i></b> – return the contents of register <b>a0</b> (also <b>a1</b> – <b>a7</b> ) (MC680x0)                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>int a0 (     int taskId /* task ID, 0 means default task */ )</pre>                                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This command extracts the contents of register <b>a0</b> from the TCB of a specified task. If <i>taskId</i> is omitted or zero, the last task referenced is assumed.</p> <p>Similar routines are provided for all address registers (<b>a0</b> – <b>a7</b>): <b><i>a0()</i></b> – <b><i>a7()</i></b>.</p> <p>The stack pointer is accessed via <b><i>a7()</i></b>.</p> |
| <b>RETURNS</b>     | The contents of register <b>a0</b> (or the requested register).                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>dbgArchLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                                                       |

---

## ***abort()***

|                      |                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b><i>abort()</i></b> – cause abnormal program termination (ANSI)                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>      | <pre>void abort (void)</pre>                                                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b>   | <p>This routine causes abnormal program termination, unless the signal <b>SIGABRT</b> is being caught and the signal handler does not return. VxWorks does not flush output streams, close open streams, or remove temporary files. <b><i>abort()</i></b> returns unsuccessful status termination to the host environment by calling:</p> <pre>raise (SIGABRT);</pre> |
| <b>INCLUDE FILES</b> | <b>stdlib.h</b>                                                                                                                                                                                                                                                                                                                                                       |
| <b>RETURNS</b>       | This routine cannot return to the caller.                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>      | <b>ansiStdlib</b>                                                                                                                                                                                                                                                                                                                                                     |

**abs()**

---

**abs()**

|                      |                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>abs()</i> – compute the absolute value of an integer (ANSI)                                                                   |
| <b>SYNOPSIS</b>      | <pre>int abs (     int i /* integer for which to return absolute value */ )</pre>                                                |
| <b>DESCRIPTION</b>   | This routine computes the absolute value of a specified integer. If the result cannot be represented, the behavior is undefined. |
| <b>INCLUDE FILES</b> | <code>stdlib.h</code>                                                                                                            |
| <b>RETURNS</b>       | The absolute value of <i>i</i> .                                                                                                 |
| <b>SEE ALSO</b>      | <code>ansiStdlib</code>                                                                                                          |

---

**accept()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>accept()</i> – accept a connection from a socket                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>int accept (     int          s,          /* socket descriptor */     struct sockaddr * addr,  /* peer address */     int *       addrLen /* peer address length */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | <p>This routine accepts a connection on a socket, and returns a new socket created for the connection. The socket must be bound to an address with <i>bind()</i>, and enabled for connections by a call to <i>listen()</i>. The <i>accept()</i> routine dequeues the first connection and creates a new socket with the same properties as <i>s</i>. It blocks the caller until a connection is present, unless the socket is marked as non-blocking.</p> <p>The parameter <i>addrLen</i> should be initialized to the size of the available buffer pointed to by <i>addr</i>. Upon return, <i>addrLen</i> contains the size in bytes of the peer's address stored in <i>addr</i>.</p> |
| <b>RETURNS</b>     | A socket descriptor, or ERROR if the call fails.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <code>sockLib</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

---

**acos()**

|                      |                                                                                                                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>acos()</i> – compute an arc cosine (ANSI)                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>      | <pre>double acos (     double x /* number between -1 and 1 */ )</pre>                                                                                                                                                                                             |
| <b>DESCRIPTION</b>   | This routine returns principal value of the arc cosine of <i>x</i> in double precision (IEEE double, 53 bits). If <i>x</i> is the cosine of an angle <i>T</i> , this function returns <i>T</i> .<br>A domain error occurs for arguments not in the range [-1,+1]. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                                                                                                                                     |
| <b>RETURNS</b>       | The double-precision arc cosine of <i>x</i> in the range [0,pi] radians.<br>Special cases:<br>If <i>x</i> is NaN, <i>acos()</i> returns <i>x</i> .<br>If $ x  > 1$ , it returns NaN.                                                                              |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                                                                                                                                                                                         |

---

**acosf()**

|                      |                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>acosf()</i> – compute an arc cosine (ANSI)                                                                                                           |
| <b>SYNOPSIS</b>      | <pre>float acosf (     float x /* number between -1 and 1 */ )</pre>                                                                                    |
| <b>DESCRIPTION</b>   | This routine computes the arc cosine of <i>x</i> in single precision. If <i>x</i> is the cosine of an angle <i>T</i> , this function returns <i>T</i> . |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                           |
| <b>RETURNS</b>       | The single-precision arc cosine of <i>x</i> in the range 0 to pi radians.                                                                               |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                                                                         |

**acw()**

---

***acw()***

**NAME** *acw()* – return the contents of the **acw** register (i960)

**SYNOPSIS**

```
int acw
(
 int taskId /* task ID, 0 means default task */
)
```

**DESCRIPTION** This command extracts the contents of the **acw** register from the TCB of a specified task. If *taskId* is omitted or 0, the current default task is assumed.

**RETURNS** The contents of the **acw** register.

**SEE ALSO** **dbgArchLib**, *VxWorks Programmer's Guide: Target Shell*

---

***aic7880CtrlCreate()***

**NAME** *aic7880CtrlCreate()* – create a control structure for the AIC 7880

**SYNOPSIS**

```
AIC_7880_SCSI_CTRL * aic7880CtrlCreate
(
 int busNo, /* PCI bus Number */
 int devNo, /* PCI device Number */
 int scsiBusId /* SCSI Host Adapter Bus Id */
)
```

**DESCRIPTION** This routine creates an **AIC\_7880\_SCSI\_CTRL** structure and must be called before using the SCSI Host Adapter chip. It must be called exactly once for a specified Host Adapter.

**RETURNS** A pointer to the **AIC\_7880\_SCSI\_CTRL** structure, or NULL if memory is unavailable or there are invalid parameters.

**SEE ALSO** **aic7880Lib**



---

## *aic7880dFifoThresholdSet()*

**NAME** *aic7880dFifoThresholdSet()* – set the data FIFO threshold.

**SYNOPSIS**

```
STATUS aic7880dFifoThresholdSet
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller */
 UBYTE threshHold /* data FIFO threshold value */
)
```

**DESCRIPTION** This routine specifies to the AIC-7880 host adapter how to manage its data FIFO. Below is a description of the threshold values for SCSI reads and writes.

**SCSI READS**

- 0 Xfer data from FIFO as soon as it is available.
- 1 Xfer data from FIFO as soon as the FIFO is half full.
- 2 Xfer data from FIFO as soon as the FIFO is 75% full.
- 3 Xfer data from FIFO as soon as the FIFO is 100% full.

**SCSI WRITES**

- 0 Xfer data as soon as there is room in the FIFO.
- 1 Xfer data to FIFO as soon as it is 50% empty.
- 2 Xfer data to FIFO as soon as it is 75% empty.
- 3 Xfer data to FIFO as soon as the FIFO is empty.

**RETURNS** OK or ERROR if the threshold value is not within the valid range.

**SEE ALSO** *aic7880Lib*

---

## *aic7880EnableFast20()*

**NAME** *aic7880EnableFast20()* – enable double speed SCSI data transfers

**SYNOPSIS**

```
VOID aic7880EnableFast20
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller */
 BOOL enable /* enable = 1 / disable = 0 */
)
```

**DESCRIPTION** This routine enables double speed SCSI data transfers for the SCSI host adapter. This allows the host adapter to transfer data upto 20 MB/s for an 8 bit device and upto 40 MB/s for a 16 bit device.

**RETURNS** N/A

**SEE ALSO** ***aic7880Lib***

---

## ***aic7880GetNumOfBuses()***

**NAME** ***aic7880GetNumOfBuses()*** – perform a PCI bus scan

**SYNOPSIS** **DWORD *aic7880GetNumOfBuses* ( )**

**DESCRIPTION** This routine provides a callback mechanism from the HIM to the OSM. It allows the OSM to scan the PCI bus, before the HIM is allowed to perform the bus scan.

**RETURNS** 0x55555555 if the OSM is not able to conduct its own bus scan

**SEE ALSO** ***aic7880Lib***

---

## ***aic7880ReadConfig()***

**NAME** ***aic7880ReadConfig()*** – read from PCI config space

**SYNOPSIS** **DWORD *aic7880ReadConfig***

```
(
 cfp_struct * configPtr, /* ptr to cf_struct */
 UBYTE busNo, /* PCI bus number */
 UBYTE devNo, /* PCI device number */
 UBYTE regNo /* register */
)
```

**DESCRIPTION** This routine provides a callback mechanism from the HIM to the OSM. The purpose of this routine is to allow the OSM to do its own Read access of the PCI configuration space. If the OSM cannot successfully complete the Read access, the OSM returns 0x55555555. If this happens the HIM attempts to conduct the configuration space Read access.

**RETURNS** value read or 0x55555555, if the OSM is not able to conduct read access to the PCI configuration space.

**SEE ALSO** ***aic7880Lib***

---

## *aic7880ScbCompleted()*

**NAME** *aic7880ScbCompleted()* – successfully completed execution of a client thread

**SYNOPSIS**

```
VOID aic7880ScbCompleted
(
 sp_struct * pScb /* ptr to completed SCSI Command Block */
)
```

**DESCRIPTION** This routine is called from within the context of the ISR. The HIM calls this routine passing in the pointer of the of the completed SCB. This routine sets the thread status, handles the completed SCB and returns program control back to the HIM which then returns from the *PH\_IntHandler()* routine.

This routine could be called more than once from the same PH\_IntHandler call. Each call to this routine indicates the completion of an SCB. For each SCB completed, this routine sets the event type and calls the appropriate AIC-7880 event handler routines which sets the SCSI Controller, SCSI Physical Device and SCSI Thread, state variables appropriately. This routine also handles synchronization with the SCSI Manager so that the next runnable thread can be scheduled for execution.

**RETURNS** N/A

**SEE ALSO** *aic7880Lib*

---

## *aic7880WriteConfig()*

**NAME** *aic7880WriteConfig()* – read to PCI config space

**SYNOPSIS**

```
DWORD aic7880WriteConfig
(
 cfp_struct * config_ptr, /* ptr to cf_struct */
 UBYTE busNo, /* PCI bus number */
 UBYTE devNo, /* PCI device number */
 UBYTE regNo, /* register */
 DWORD regVal /* register value */
)
```

**DESCRIPTION** This routine provides a callback mechanism from the HIM to the OSM. The purpose of this routine is to allow the OSM to do its own write access of the PCI configuration space.

If the OSM cannot successfully complete the write access, the OSM returns 0x55555555. If this happens the HIM attempts to conduct the configuration space write access.

**RETURNS** OK or 0x55555555, if the OSM is not able to conduct write access to the PCI configuration space.

**SEE ALSO** ***aic7880Lib***

---

## ***aioPxLibInit()***

**NAME** ***aioPxLibInit()*** – initialize the asynchronous I/O (AIO) library

**SYNOPSIS**

```
STATUS aioPxLibInit
(
 int lioMax /* max outstanding lio calls */
)
```

**DESCRIPTION** This routine initializes the AIO library. It should be called only once after the I/O system has been initialized. *lioMax* specifies the maximum number of outstanding *lio\_listio()* calls at one time. If *lioMax* is zero, the default value of **AIO\_CLUST\_MAX** is used.

**RETURNS** OK if successful, otherwise ERROR.

**ERRNO** **S\_***aioPxLib\_IOS\_NOT\_INITIALIZED*

**SEE ALSO** ***aioPxLib***

---

## ***aioShow()***

**NAME** ***aioShow()*** – show AIO requests

**SYNOPSIS**

```
STATUS aioShow
(
 int drvNum /* drv num to show (IGNORED) */
)
```

**DESCRIPTION** This routine displays the outstanding AIO requests.

**CAVEAT** The *drvNum* parameter is not currently used.

**RETURNS** OK, always.

**SEE ALSO** `aioPxShow`

---

## *aioSysInit()*

**NAME** *aioSysInit()* – initialize the AIO system driver

**SYNOPSIS**

```
STATUS aioSysInit
(
 int numTasks, /* number of system tasks */
 int taskPrio, /* AIO task priority */
 int taskStackSize /* AIO task stack size */
)
```

**DESCRIPTION** This routine initializes the AIO system driver. It should be called once after the AIO library has been initialized. It spawns *numTasks* system I/O tasks to be executed at *taskPrio* priority level, with a stack size of *taskStackSize*. It also starts the wait task and sets the system driver as the default driver for AIO. If *numTasks*, *taskPrio*, or *taskStackSize* is 0, a default value (`AIO_IO_TASKS_DFLT`, `AIO_IO_PRIO_DFLT`, or `AIO_IO_STACK_DFLT`, respectively) is used.

**RETURNS** OK if successful, otherwise ERROR.

**SEE ALSO** `aioSysDrv`

---

## *aio\_error()*

**NAME** *aio\_error()* – retrieve error status of asynchronous I/O operation (POSIX)

**SYNOPSIS**

```
int aio_error
(
 const struct aiocb * pAiocb /* AIO control block */
)
```

**DESCRIPTION** This routine returns the error status associated with the I/O operation specified by *pAiocb*. If the operation is not yet completed, the error status will be `EINPROGRESS`.

***aio\_fsync()***

|                      |                                                                                                                                                                        |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>RETURNS</b>       | EINPROGRESS if the AIO operation has not yet completed, OK if the AIO operation completed successfully, the error status if the AIO operation failed, otherwise ERROR. |
| <b>ERRNO</b>         | EINVAL                                                                                                                                                                 |
| <b>INCLUDE FILES</b> | aio.h                                                                                                                                                                  |
| <b>SEE ALSO</b>      | aioPxLib                                                                                                                                                               |

---

## ***aio\_fsync()***

**NAME** *aio\_fsync()* – asynchronous file synchronization (POSIX)

**SYNOPSIS**

```
int aio_fsync
(
 int op, /* operation */
 struct aiocb * pAiocb /* AIO control block */
)
```

**DESCRIPTION** This routine asynchronously forces all I/O operations associated with the file, indicated by **aio\_fildes**, queued at the time *aio\_fsync()* is called to the synchronized I/O completion state. *aio\_fsync()* returns when the synchronization request has been initiated or queued to the file or device.

The value of *op* is ignored. It currently has no meaning in VxWorks.

If the call fails, the outstanding I/O operations are not guaranteed to have completed. If it succeeds, only the I/O that was queued at the time of the call is guaranteed to the relevant completion state.

The **aio\_sigevent** member of the *pAiocb* defines an optional signal to be generated on completion of *aio\_fsync()*.

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <b>RETURNS</b>       | OK if queued successfully, otherwise ERROR.        |
| <b>ERRNO</b>         | EINVAL, EBADF                                      |
| <b>INCLUDE FILES</b> | aio.h                                              |
| <b>SEE ALSO</b>      | aioPxLib, <i>aio_error()</i> , <i>aio_return()</i> |

---

## *aio\_read( )*

**NAME**  *aio\_read( )* – initiate an asynchronous read (POSIX)

**SYNOPSIS**

```
int aio_read
(
 struct aiocb * pAiocb /* AIO control block */
)
```

**DESCRIPTION** This routine asynchronously reads data based on the following parameters specified by members of the AIO control structure  *pAiocb*. It reads  **aio\_nbytes** bytes of data from the file  **aio\_fildes** into the buffer  **aio\_buf**.

The requested operation takes place at the absolute position in the file as specified by  **aio\_offset**.

**aio\_reqprio** can be used to lower the priority of the AIO request; if this parameter is nonzero, the priority of the AIO request is  **aio\_reqprio** lower than the calling task priority.

The call returns when the read request has been initiated or queued to the device.  *aio\_error( )* can be used to determine the error status and of the AIO operation. On completion,  *aio\_return( )* can be used to determine the return status.

**aio\_sigevent** defines the signal to be generated on completion of the read request. If this value is zero, no signal is generated.

**RETURNS** OK if the read queued successfully, otherwise ERROR.

**ERRNO** EBAADF, EINVAL

**INCLUDE FILES**  *aio.h*

**SEE ALSO**  *aioPxLib, aio\_error( ), aio\_return( ), read( )*

---

## *aio\_return( )*

**NAME**  *aio\_return( )* – retrieve return status of asynchronous I/O operation (POSIX)

**SYNOPSIS**

```
size_t aio_return
(
 struct aiocb * pAiocb /* AIO control block */
)
```

***aio\_suspend()***

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b>   | This routine returns the return status associated with the I/O operation specified by <i>pAiocb</i> . The return status for an AIO operation is the value that would be returned by the corresponding <i>read()</i> , <i>write()</i> , or <i>fsync()</i> call. <i>aio_return()</i> may be called only after the AIO operation has completed ( <i>aio_error()</i> returns a valid error code--not <b>EINPROGRESS</b> ). Furthermore, <i>aio_return()</i> may be called only once; subsequent calls will fail. |
| <b>RETURNS</b>       | The return status of the completed AIO request, or <b>ERROR</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>ERRNO</b>         | <b>EINVAL</b> , <b>EINPROGRESS</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>INCLUDE FILES</b> | <b>aio.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>      | <b>aioPxLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

---

## ***aio\_suspend()***

|                      |                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>aio_suspend()</i> – wait for asynchronous I/O request(s) (POSIX)                                                                                                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>      | <pre>int aio_suspend (     const struct aiocb *   list[], /* AIO requests */     int                   nEnt,   /* number of requests */     const struct timespec * timeout /* wait timeout */ )</pre>                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>   | <p>This routine suspends the caller until one of the following occurs:</p> <ul style="list-style-type: none"> <li>– at least one of the previously submitted asynchronous I/O operations referenced by <i>list</i> has completed,</li> <li>– a signal interrupts the function, or</li> <li>– the time interval specified by <i>timeout</i> has passed (if <i>timeout</i> is not <b>NULL</b>).</li> </ul> |
| <b>RETURNS</b>       | OK if an AIO request completes, otherwise <b>ERROR</b> .                                                                                                                                                                                                                                                                                                                                                 |
| <b>ERRNO</b>         | <b>EAGAIN</b> , <b>EINTR</b>                                                                                                                                                                                                                                                                                                                                                                             |
| <b>INCLUDE FILES</b> | <b>aio.h</b>                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>      | <b>aioPxLib</b>                                                                                                                                                                                                                                                                                                                                                                                          |



---

## *aio\_write()*

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>aio_write()</i> – initiate an asynchronous write (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>      | <pre>int aio_write (     struct aiocb * pAiocb /* AIO control block */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b>   | <p>This routine asynchronously writes data based on the following parameters specified by members of the AIO control structure <i>pAiocb</i>. It writes <b>aio_nbytes</b> of data to the file <b>aio_fildes</b> from the buffer <b>aio_buf</b>.</p> <p>The requested operation takes place at the absolute position in the file as specified by <b>aio_offset</b>.</p> <p><b>aio_reqprio</b> can be used to lower the priority of the AIO request; if this parameter is nonzero, the priority of the AIO request is <b>aio_reqprio</b> lower than the calling task priority.</p> <p>The call returns when the write request has been initiated or queued to the device. <i>aio_error()</i> can be used to determine the error status and of the AIO operation. On completion, <i>aio_return()</i> can be used to determine the return status.</p> <p><b>aio_sigevent</b> defines the signal to be generated on completion of the write request. If this value is zero, no signal is generated.</p> |
| <b>RETURNS</b>       | OK if write queued successfully, otherwise ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>ERRNO</b>         | EBADEF, EINVAL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>INCLUDE FILES</b> | <b>aio.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>      | <b>aioPxBLib</b> , <i>aio_error()</i> , <i>aio_return()</i> , <i>writel()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

---

## *ambaDevInit()*

|                 |                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <i>ambaDevInit()</i> – initialise an AMBA channel                                                    |
| <b>SYNOPSIS</b> | <pre>void ambaDevInit (     AMBA_CHAN * pChan /* ptr to AMBA_CHAN describing this channel */ )</pre> |

## ***ambaIntRx()***

|                    |                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine initialises some <b>SIO_CHAN</b> function pointers and then resets the chip to a quiescent state. Before this routine is called, the BSP must already have initialised all the device addresses, etc. in the <b>AMBA_CHAN</b> structure. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <b>ambaSio</b>                                                                                                                                                                                                                                        |

---

## ***ambaIntRx()***

|                    |                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ambaIntRx()</i> – handle a receiver interrupt                                                   |
| <b>SYNOPSIS</b>    | <pre>void ambaIntRx (     AMBA_CHAN * pChan /* ptr to AMBA_CHAN describing this channel */ )</pre> |
| <b>DESCRIPTION</b> | This routine handles read interrupts from the UART.                                                |
| <b>RETURNS</b>     | N/A                                                                                                |
| <b>SEE ALSO</b>    | <b>ambaSio</b>                                                                                     |

---

## ***ambaIntTx()***

|                    |                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ambaIntTx()</i> – handle a transmitter interrupt                                                |
| <b>SYNOPSIS</b>    | <pre>void ambaIntTx (     AMBA_CHAN * pChan /* ptr to AMBA_CHAN describing this channel */ )</pre> |
| <b>DESCRIPTION</b> | This routine handles write interrupts from the UART.                                               |
| <b>RETURNS</b>     | N/A                                                                                                |
| <b>SEE ALSO</b>    | <b>ambaSio</b>                                                                                     |

---

## *arpAdd()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>arpAdd()</i> – add an entry to the system ARP table                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>STATUS arpAdd (     char * host, /* host name or IP address */     char * eaddr, /* Ethernet address */     int flags /* ARP flags */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This routine adds a specified entry to the ARP table. <i>host</i> is a valid host name or Internet address. <i>eaddr</i> is the Ethernet address of the host and has the form "x:x:x:x:x" where x is a hexadecimal number between 0 and ff.</p> <p>The <i>flags</i> parameter specifies the ARP flags for the entry; the following bits are settable:</p> <p><b>ATF_PERM</b> (0x04)<br/>The ATF_PERM bit makes the ARP entry permanent. A permanent ARP entry does not time out as do normal ARP entries.</p> <p><b>ATF_PUBL</b> (0x08)<br/>The ATF_PUBL bit causes the entry to be published (i.e., this system responds to ARP requests for this entry, even though it is not the host).</p> <p><b>ATF_USETRAILERS</b> (0x10)<br/>The ATF_USETRAILERS bit indicates that trailer encapsulations can be sent to this host.</p> |
| <b>EXAMPLE</b>     | <p>* The following call creates a permanent ARP table entry for the host with IP address 90.0.0.3 and Ethernet address 0:80:f9:1:2:3:</p> <pre>arpAdd ("90.0.0.3", "0:80:f9:1:2:3", 0x4)</pre> <p>The following call adds an entry to the ARP table for host "myHost", with an Ethernet address of 0:80:f9:1:2:4; no flags are set for this entry:</p> <pre>arpAdd ("myHost", "0:80:f9:1:2:4", 0)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>RETURNS</b>     | OK, or ERROR if unsuccessful.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>ERRNO</b>       | S_arpLib_INVALID_ARGUMENT, S_arpLib_INVALID_FLAG                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | arpLib                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

## ***arpDelete()***

|                    |                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>arpDelete()</i> – delete an entry from the system ARP table                                                                   |
| <b>SYNOPSIS</b>    | <pre>STATUS arpDelete (     char * host /* host name or IP address */ )</pre>                                                    |
| <b>DESCRIPTION</b> | This routine deletes an ARP table entry. <i>host</i> specifies the entry to delete and is a valid host name or Internet address. |
| <b>EXAMPLE</b>     | <pre>arpDelete ("91.0.0.3") arpDelete ("myHost")</pre>                                                                           |
| <b>RETURNS</b>     | OK, or ERROR if unsuccessful.                                                                                                    |
| <b>ERRNO</b>       | S_arpLib_INVALID_ARGUMENT                                                                                                        |
| <b>SEE ALSO</b>    | <b>arpLib</b>                                                                                                                    |

---

## ***arpFlush()***

|                    |                                                                  |
|--------------------|------------------------------------------------------------------|
| <b>NAME</b>        | <i>arpFlush()</i> – flush all entries in the system ARP table    |
| <b>SYNOPSIS</b>    | <pre>void arpFlush (void)</pre>                                  |
| <b>DESCRIPTION</b> | This routine flushes all non-permanent entries in the ARP cache. |
| <b>RETURNS</b>     | N/A                                                              |
| <b>SEE ALSO</b>    | <b>arpLib</b>                                                    |

---

## *arpShow()*

**NAME** *arpShow()* – display entries in the system ARP table

**SYNOPSIS** `void arpShow (void)`

**DESCRIPTION** This routine displays the current Internet-to-Ethernet address mappings in the ARP table.

**EXAMPLE**

```
-> arpShow
LINK LEVEL ARP TABLE
destination gateway flags Refcnt Use Interface

90.0.0.63 08:00:3e:23:79:e7 405 0 82 lo0

```

**RETURNS** N/A

**SEE ALSO** `netShow`

---

## *arptabShow()*

**NAME** *arptabShow()* – display the known ARP entries

**SYNOPSIS** `void arptabShow (void)`

**DESCRIPTION** This routine displays current Internet-to-Ethernet address mappings in the ARP table.

**RETURNS** N/A

**SEE ALSO** `netShow`

## ***asctime()***

- NAME** *asctime()* – convert broken-down time into a string (ANSI)
- SYNOPSIS**
- ```
char * asctime
(
    const struct tm * timeptr /* broken-down time */
)
```
- DESCRIPTION** This routine converts the broken-down time pointed to by *timeptr* into a string of the form:
- ```
SUN SEP 16 01:03:52 1973\n\0
```
- This routine is not reentrant. For a reentrant version, see *asctime\_r()*.
- INCLUDE FILES** **time.h**
- RETURNS** A pointer to the created string.
- SEE ALSO** **ansiTime**

---

## ***asctime\_r()***

- NAME** *asctime\_r()* – convert broken-down time into a string (POSIX)
- SYNOPSIS**
- ```
int asctime_r
(
    const struct tm * timeptr, /* broken-down time */
    char *          asctimeBuf, /* buffer to contain string */
    size_t *        buflen     /* size of buffer */
)
```
- DESCRIPTION** This routine converts the broken-down time pointed to by *timeptr* into a string of the form:
- ```
SUN SEP 16 01:03:52 1973\n\0
```
- The string is copied to *asctimeBuf*. This call is the POSIX re-entrant version of *asctime()*.
- INCLUDE FILES** **time.h**
- RETURNS** The size of the created string.
- SEE ALSO** **ansiTime**

---

***asin()***

|                      |                                                                                                                                                                                                                                                  |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>asin()</i> – compute an arc sine (ANSI)                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>      | <pre>double asin (     double x /* number between -1 and 1 */ )</pre>                                                                                                                                                                            |
| <b>DESCRIPTION</b>   | This routine returns the principal value of the arc sine of $x$ in double precision (IEEE double, 53 bits). If $x$ is the sine of an angle $T$ , this function returns $T$ .<br>A domain error occurs for arguments not in the range $[-1,+1]$ . |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                                                                                                                    |
| <b>RETURNS</b>       | The double-precision arc sine of $x$ in the range $[-\pi/2,\pi/2]$ radians.<br>Special cases:<br>If $x$ is NaN, <i>asin()</i> returns $x$ .<br>If $ x >1$ , it returns NaN.                                                                      |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                                                                                                                                                                        |

---

***asinf()***

|                      |                                                                                                                                 |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>asinf()</i> – compute an arc sine (ANSI)                                                                                     |
| <b>SYNOPSIS</b>      | <pre>float asinf (     float x /* number between -1 and 1 */ )</pre>                                                            |
| <b>DESCRIPTION</b>   | This routine computes the arc sine of $x$ in single precision. If $x$ is the sine of an angle $T$ , this function returns $T$ . |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                   |
| <b>RETURNS</b>       | The single-precision arc sine of $x$ in the range $-\pi/2$ to $\pi/2$ radians.                                                  |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                                                 |

**assert()**

---

**assert()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>assert()</i> – put diagnostics into programs (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>void assert (     int a )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>If an expression is false (that is, equal to zero), the <i>assert()</i> macro writes information about the failed call to standard error in an implementation-defined format. It then calls <i>abort()</i>. The diagnostic information includes:</p> <ul style="list-style-type: none"><li>– the text of the argument</li><li>– the name of the source file (value of preprocessor macro <code>__FILE__</code>)</li><li>– the source line number (value of preprocessor macro <code>__LINE__</code>)</li></ul> |
| <b>INCLUDE</b>     | <code>stdio.h</code> , <code>stdlib.h</code> , <code>assert.h</code>                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <code>ansiAssert</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

---

**ataDevCreate()**

|                    |                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ataDevCreate()</i> – create a device for a ATA/IDE disk                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>BLK_DEV *ataDevCreate (     int ctrl,     int drive,     int nBlocks,     int blkOffset )</pre>                                                                                                                                                                               |
| <b>DESCRIPTION</b> | <p>This routine creates a device for a specified ATA/IDE disk.</p> <p><i>drive</i> is a drive number for the hard drive; it must be 0 or 1.</p> <p>The <i>nBlocks</i> parameter specifies the size of the device in blocks. If <i>nBlocks</i> is zero, the whole disk is used.</p> |



The *blkOffset* parameter specifies an offset, in blocks, from the start of the device to be used when writing or reading the hard disk. This offset is added to the block numbers passed by the file system during disk accesses. (VxWorks file systems always use block numbers beginning at zero for the start of a device.)

**RETURNS** A pointer to a block device structure (**BLK\_DEV**) or **NULL** if memory cannot be allocated for the device structure.

**SEE ALSO** *ataDrv*, *dosFsMkfs()*, *dosFsDevInit()*, *rt11FsDevInit()*, *rt11FsMkfs()*, *rawFsDevInit()*

---

## *ataDrv()*

**NAME** *ataDrv()* – initialize the ATA driver

**SYNOPSIS**

```
STATUS ataDrv
(
 int ctrl, /* controller no. */
 int drives, /* number of drives */
 int vector, /* interrupt vector */
 int level, /* interrupt level */
 BOOL configType, /* configuration type */
 int semTimeout, /* timeout seconds for sync semaphore */
 int wdgTimeout /* timeout seconds for watch dog */
)
```

**DESCRIPTION** This routine initializes the ATA/IDE driver, sets up interrupt vectors, and performs hardware initialization of the ATA/IDE chip.

This routine must be called exactly once, before any reads, writes, or calls to *ataDevCreate()*. Normally, it is called by *usrRoot()* in *usrConfig.c*.

**RETURNS** **OK**, or **ERROR** if initialization fails.

**SEE ALSO** *ataDrv*, *ataDevCreate()*

***atan()***

---

***atan()***

|                      |                                                                                                                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>atan()</i> – compute an arc tangent (ANSI)                                                                                                                                                  |
| <b>SYNOPSIS</b>      | <pre>double atan (     double x /* tangent of an angle */ )</pre>                                                                                                                              |
| <b>DESCRIPTION</b>   | This routine returns the principal value of the arc tangent of $x$ in double precision (IEEE double, 53 bits). If $x$ is the tangent of an angle $T$ , this function returns $T$ (in radians). |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                                                                  |
| <b>RETURNS</b>       | The double-precision arc tangent of $x$ in the range $[-\pi/2, \pi/2]$ radians. Special case: if $x$ is NaN, <i>atan()</i> returns $x$ itself.                                                 |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                                                                                                                      |

---

***atan2()***

|                      |                                                                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>atan2()</i> – compute the arc tangent of $y/x$ (ANSI)                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>      | <pre>double atan2 (     double y, /* numerator */     double x /* denominator */ )</pre>                                                                                                                                                                           |
| <b>DESCRIPTION</b>   | This routine returns the principal value of the arc tangent of $y/x$ in double precision (IEEE double, 53 bits). This routine uses the signs of both arguments to determine the quadrant of the return value. A domain error may occur if both arguments are zero. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                                                                                                                                      |
| <b>RETURNS</b>       | The double-precision arc tangent of $y/x$ , in the range $[-\pi, \pi]$ radians.<br>Special cases:<br>Notations: $\text{atan2}(y,x) == \text{ARG}(x+iy) == \text{ARG}(x,y)$ .<br>$\text{ARG}(\text{NaN}, (\text{anything}))$ is NaN                                 |

|                                           |    |         |
|-------------------------------------------|----|---------|
| ARG(anything), NaN                        | is | NaN     |
| ARG+(anything but NaN), +-0               | is | +-0     |
| ARG-(anything but NaN), +-0               | is | +-PI    |
| ARG(0, ++(anything but 0 and NaN))        | is | +-PI/2  |
| ARG(+INF, ++(anything but INF and NaN))   | is | +-0     |
| ARG(-INF, ++(anything but INF and NaN))   | is | +-PI    |
| ARG(+INF, +-INF)                          | is | +-PI/4  |
| ARG(-INF, +-INF)                          | is | +-3PI/4 |
| ARG((anything but 0, NaN, and INF),+-INF) | is | +-PI/2  |

SEE ALSO **ansiMath, mathALib**

## *atan2f()*

**NAME** *atan2f()* – compute the arc tangent of  $y/x$  (ANSI)

**SYNOPSIS**

```
float atan2f
(
 float y, /* numerator */
 float x /* denominator */
)
```

**DESCRIPTION** This routine returns the principal value of the arc tangent of  $y/x$  in single precision.

**INCLUDE FILES** **math.h**

**RETURNS** The single-precision arc tangent of  $y/x$  in the range  $-\pi$  to  $\pi$ .

**SEE ALSO** **mathALib**

## *atanf()*

**NAME** *atanf()* – compute an arc tangent (ANSI)

**SYNOPSIS**

```
float atanf
(
 float x /* tangent of an angle */
)
```

**DESCRIPTION** This routine computes the arc tangent of  $x$  in single precision. If  $x$  is the tangent of an angle  $T$ , this function returns  $T$  (in radians).

**INCLUDE FILES** **math.h**

**RETURNS** The single-precision arc tangent of  $x$  in the range  $-\pi/2$  to  $\pi/2$ .

**SEE ALSO** **mathALib**

---

## ***ataRawio()***

**NAME** ***ataRawio()*** – do raw I/O access

**SYNOPSIS**

```
STATUS ataRawio
(
 int ctrl,
 int drive,
 ATA_RAW * pAtaRaw
)
```

**DESCRIPTION** This routine is called to perform raw I/O access.  
*drive* is a drive number for the hard drive: it must be 0 or 1.  
The *pAtaRaw* is a pointer to the structure **ATA\_RAW** which is defined in **ataDrv.h**.

**RETURNS** OK, or ERROR if the parameters are not valid.

**SEE ALSO** **ataDrv**

---

## ***ataShow()***

**NAME** ***ataShow()*** – show the ATA/IDE disk parameters

**SYNOPSIS**

```
STATUS ataShow
(
 int ctrl,
 int drive
)
```

|                    |                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine shows the ATA/IDE disk parameters. Its first argument is a controller number, 0 or 1; the second argument is a drive number, 0 or 1. |
| <b>RETURNS</b>     | OK, or ERROR if the parameters are invalid.                                                                                                       |
| <b>SEE ALSO</b>    | <b>ataShow</b>                                                                                                                                    |

---

## *ataShowInit()*

|                    |                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ataShowInit()</i> – initialize the ATA/IDE disk driver show routine                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <code>void ataShowInit (void)</code>                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | This routine links the ATA/IDE disk driver show routine into the VxWorks system. The routine is included automatically by defining <code>INCLUDE_SHOW_ROUTINES</code> in <code>configAll.h</code> . No arguments are needed. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>ataShow</b>                                                                                                                                                                                                               |

---

## *atexit()*

|                      |                                                                                    |
|----------------------|------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>atexit()</i> – call a function at program termination (Unimplemented) (ANSI)    |
| <b>SYNOPSIS</b>      | <pre>int atexit (     void (* __func)(void) /* pointer to a function */ )</pre>    |
| <b>DESCRIPTION</b>   | This routine is unimplemented. VxWorks task exit hooks provide this functionality. |
| <b>INCLUDE FILES</b> | <code>stdlib.h</code>                                                              |
| <b>RETURNS</b>       | ERROR, always.                                                                     |
| <b>SEE ALSO</b>      | <b>ansiStdlib, taskHookLib</b>                                                     |

**atof()**

---

**atof()**

**NAME**            *atof()* – convert a string to a **double** (ANSI)

**SYNOPSIS**        `double atof`  
                  (  
                  `const char * s /* pointer to string */`  
                  )

**DESCRIPTION**    This routine converts the initial portion of the string *s* to double-precision representation. Its behavior is equivalent to:

`strtod (s, (char **)NULL);`

**INCLUDE FILES**   `stdlib.h`

**RETURNS**         The converted value in double-precision representation.

**SEE ALSO**         `ansiStdlib`

---

**atoi()**

**NAME**            *atoi()* – convert a string to an **int** (ANSI)

**SYNOPSIS**        `int atoi`  
                  (  
                  `const char * s /* pointer to string */`  
                  )

**DESCRIPTION**    This routine converts the initial portion of the string *s* to **int** representation. Its behavior is equivalent to:

`(int) strtol (s, (char **) NULL, 10);`

**INCLUDE FILES**   `stdlib.h`

**RETURNS**         The converted value represented as an **int**.

**SEE ALSO**         `ansiStdlib`

---

## *atol()*

|                      |                                                                                                                                                                              |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>atol()</i> – convert a string to a <b>long</b> (ANSI)                                                                                                                     |
| <b>SYNOPSIS</b>      | <pre>long atol (     const register char * s /* pointer to string */ )</pre>                                                                                                 |
| <b>DESCRIPTION</b>   | This routine converts the initial portion of the string <i>s</i> to long integer representation. Its behavior is equivalent to:<br><pre>strtol (s, (char **)NULL, 10);</pre> |
| <b>INCLUDE FILES</b> | <code>stdlib.h</code>                                                                                                                                                        |
| <b>RETURNS</b>       | The converted value represented as a <b>long</b> .                                                                                                                           |
| <b>SEE ALSO</b>      | <code>ansiStdlib</code>                                                                                                                                                      |

---

## *b()*

|                    |                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>b()</i> – set or display breakpoints                                                                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>STATUS b (     INSTR * addr, /* where to set breakpoint, 0 = display all breakpoints */     int     task, /* task for which to set breakpoint, 0 = set all tasks */     int     count, /* number of passes before hit */     BOOL    quiet /* TRUE = don't print debugging info, FALSE = print */               /* info */ )</pre> |
| <b>DESCRIPTION</b> | This routine sets or displays breakpoints. To display the list of currently active breakpoints, call <i>b()</i> without arguments:<br><pre>-&gt; b</pre> <p>The list shows the address, task, and pass count of each breakpoint. Temporary breakpoints inserted by <i>so()</i> and <i>cret()</i> are also indicated.</p>                |

***bcmp()***

To set a breakpoint with *b()*, include the address, which can be specified numerically or symbolically with an optional offset. The other arguments are optional:

```
-> b addr[,task[,count[,quiet]]]
```

If *task* is zero or omitted, the breakpoint will apply to all breakable tasks. If *count* is zero or omitted, the breakpoint will occur every time it is hit. If *count* is specified, the break will not occur until the *count* +1th time an eligible task hits the breakpoint (i.e., the breakpoint is ignored the first *count* times it is hit).

If *quiet* is specified, debugging information destined for the console will be suppressed when the breakpoint is hit. This option is included for use by external source code debuggers that handle the breakpoint user interface themselves.

Individual tasks can be unbreakable, in which case breakpoints that otherwise would apply to a task are ignored. Tasks can be spawned unbreakable by specifying the task option `VX_UNBREAKABLE`. Tasks can also be set unbreakable or breakable by resetting `VX_UNBREAKABLE` with the routine *taskOptionsSet()*.

**RETURNS** OK, or ERROR if *addr* is illegal or the breakpoint table is full.

**SEE ALSO** *dbgLib*, *bd()*, *taskOptionsSet()*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

***bcmp()***

**NAME** *bcmp()* – compare one buffer to another

**SYNOPSIS**

```
int bcmp
(
 char * buf1, /* pointer to first buffer */
 char * buf2, /* pointer to second buffer */
 int nbytes /* number of bytes to compare */
)
```

**DESCRIPTION** This routine compares the first *nbytes* characters of *buf1* to *buf2*.

**RETURNS** 0 if the first *nbytes* of *buf1* and *buf2* are identical,  
less than 0 if *buf1* is less than *buf2*, or  
greater than 0 if *buf1* is greater than *buf2*.

**SEE ALSO** *bLib*



---

## *bcopy()*

**NAME** *bcopy()* – copy one buffer to another

**SYNOPSIS**

```
void bcopy
(
 const char * source, /* pointer to source buffer */
 char * destination, /* pointer to destination buffer */
 int nbytes /* number of bytes to copy */
)
```

**DESCRIPTION** This routine copies the first *nbytes* characters from *source* to *destination*. Overlapping buffers are handled correctly. Copying is done in the most efficient way possible, which may include long-word, or even multiple-long-word moves on some architectures. In general, the copy will be significantly faster if both buffers are long-word aligned. (For copying that is restricted to byte, word, or long-word moves, see the manual entries for *bcopyBytes()*, *bcopyWords()*, and *bcopyLongs()*.)

**RETURNS** N/A

**SEE ALSO** *bLib*, *bcopyBytes()*, *bcopyWords()*, *bcopyLongs()*

---

## *bcopyBytes()*

**NAME** *bcopyBytes()* – copy one buffer to another one byte at a time

**SYNOPSIS**

```
void bcopyBytes
(
 char * source, /* pointer to source buffer */
 char * destination, /* pointer to destination buffer */
 int nbytes /* number of bytes to copy */
)
```

**DESCRIPTION** This routine copies the first *nbytes* characters from *source* to *destination* one byte at a time. This may be desirable if a buffer can only be accessed with byte instructions, as in certain byte-wide memory-mapped peripherals.

**RETURNS** N/A

**SEE ALSO** *bLib*, *bcopy()*

---

## ***bcopyDoubles()***

**NAME** *bcopyDoubles()* – copy one buffer to another eight bytes at a time (SPARC)

**SYNOPSIS**

```
STATUS bcopyDoubles
(
 void * source, /* 8-byte aligned source buffer */
 void * destination, /* 8-byte aligned destination buffer */
 int ndoubles /* Number of 256-byte quantities */
)
```

**DESCRIPTION** This function copies the buffer *source* to the buffer *destination*, both of which must be 8-byte aligned. The copying is done eight bytes at a time. Note the count is the number of doubles, or the number of bytes divided by eight. The number of bytes copied will always be a multiple of 256.

**RETURNS** OK, if it runs to completion.

**SEE ALSO** **bALib**, *bcopy()*

---

## ***bcopyLongs()***

**NAME** *bcopyLongs()* – copy one buffer to another one long word at a time

**SYNOPSIS**

```
void bcopyLongs
(
 char * source, /* pointer to source buffer */
 char * destination, /* pointer to destination buffer */
 int nlongs /* number of longs to copy */
)
```

**DESCRIPTION** This routine copies the first *nlongs* characters from *source* to *destination* one long word at a time. This may be desirable if a buffer can only be accessed with long instructions, as in certain long-word-wide memory-mapped peripherals. The source and destination must be long-aligned.

**RETURNS** N/A

**SEE ALSO** **bLib**, *bcopy()*

---

## *bcopyWords()*

|                    |                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>bcopyWords()</i> – copy one buffer to another one word at a time                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>void bcopyWords (     char * source,      /* pointer to source buffer */     char * destination, /* pointer to destination buffer */     int   nwords       /* number of words to copy */ )</pre>                                                                                        |
| <b>DESCRIPTION</b> | This routine copies the first <i>nwords</i> words from <i>source</i> to <i>destination</i> one word at a time. This may be desirable if a buffer can only be accessed with word instructions, as in certain word-wide memory-mapped peripherals. Source and destination must be word-aligned. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>bLib</i> , <i>bcopy()</i>                                                                                                                                                                                                                                                                  |

---

## *bd()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>bd()</i> – delete a breakpoint                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS bd (     INSTR * addr, /* address of breakpoint to delete */     int   task /* task to delete breakpoint for, 0 = delete for all */ )</pre>                                                                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b> | <p>This routine deletes a specified breakpoint. To execute, enter:</p> <pre>-&gt; bd addr [,task]</pre> <p>If <i>task</i> is omitted or zero, the breakpoint will be removed for all tasks. If the breakpoint applies to all tasks, removing it for only a single task will be ineffective. It must be removed for all tasks and then set for just those tasks desired. Temporary breakpoints inserted by the routines <i>so()</i> or <i>cret()</i> can also be deleted.</p> |
| <b>RETURNS</b>     | OK, or ERROR if there is no breakpoint at the specified address.                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <i>dbgLib</i> , <i>b()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , <i>windsh</i> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                                                                                                                                                            |

***bdall()***

---

***bdall()***

|                    |                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>bdall()</i> – delete all breakpoints                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>STATUS bdall (     int task /* task for which to delete breakpoints, 0 = delete for all */ )</pre>                                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | <p>This routine removes all breakpoints. To execute, enter:</p> <pre>-&gt; bdall [task]</pre> <p>If <i>task</i> is specified, all breakpoints that apply to that task are removed. If <i>task</i> is omitted, all breakpoints for all tasks are removed. Temporary breakpoints inserted by <i>so()</i> or <i>cret()</i> are not deleted; use <i>bd()</i> instead.</p> |
| <b>RETURNS</b>     | OK, always.                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>dbgLib</b> , <i>bd()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                                                    |

---

***bfill()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>bfill()</i> – fill a buffer with a specified character                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>void bfill (     char * buf, /* pointer to buffer */     int nbytes, /* number of bytes to fill */     int ch /* char with which to fill buffer */ )</pre>                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | <p>This routine fills the first <i>nbytes</i> characters of a buffer with the character <i>ch</i>. Filling is done in the most efficient way possible, which may be long-word, or even multiple-long-word stores, on some architectures. In general, the fill will be significantly faster if the buffer is long-word aligned. (For filling that is restricted to byte stores, see the manual entry for <i>bfillBytes()</i>.)</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <b>bLib</b> , <i>bfillBytes()</i>                                                                                                                                                                                                                                                                                                                                                                                                 |

---

## ***bfillBytes()***

**NAME** *bfillBytes()* – fill buffer with a specified character one byte at a time

**SYNOPSIS**

```
void bfillBytes
(
 char * buf, /* pointer to buffer */
 int nbytes, /* number of bytes to fill */
 int ch /* char with which to fill buffer */
)
```

**DESCRIPTION** This routine fills the first *nbytes* characters of the specified buffer with the character *ch* one byte at a time. This may be desirable if a buffer can only be accessed with byte instructions, as in certain byte-wide memory-mapped peripherals.

**RETURNS** N/A

**SEE ALSO** *bLib*, *bfill()*

---

## ***bfillDoubles()***

**NAME** *bfillDoubles()* – fill a buffer with a specified eight-byte pattern (SPARC)

**SYNOPSIS**

```
STATUS bfillDoubles
(
 void * buffer, /* 8-byte aligned buffer */
 int nbytes, /* Multiple of 256 bytes */
 ULONG bits_63to32, /* Upper 32 bits of fill pattern */
 ULONG bits_31to0 /* Lower 32 bits of fill pattern */
)
```

**DESCRIPTION** This function copies a specified 8-byte pattern to the buffer, which must be 8-byte aligned. The filling is done eight bytes at a time. The number of bytes filled will be rounded up to a multiple of 256 bytes.

**RETURNS** OK, if it runs to completion.

**SEE ALSO** *bALib*, *bfill()*

**bh()**

---

**bh()**

|                    |                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>bh()</i> – set a hardware breakpoint                                                                                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre> STATUS bh (     INSTR * addr, /* where to set breakpoint, or 0 = display all */     int    access, /* access type (arch dependant) */     int    task, /* task for which to set breakpoint, 0 = set all tasks */     int    count, /* number of passes before hit */     BOOL   quiet /* TRUE = don't print debug info, FALSE = print info */ ) </pre> |
| <b>DESCRIPTION</b> | This routine is used to set a hardware breakpoint. If the architecture allows it, this function will add the breakpoint to the list of breakpoints and set the hardware breakpoint register(s). For more information, see the manual entry for <i>b()</i> .                                                                                                  |
| <b>NOTE</b>        | The types of hardware breakpoints vary with the architectures. Generally, a hardware breakpoint can be a data breakpoint or an instruction breakpoint.                                                                                                                                                                                                       |
| <b>RETURNS</b>     | OK, or ERROR if <i>addr</i> is illegal or the hardware breakpoint table is full.                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <i>dbgLib</i> , <i>b()</i> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                                 |

---

**bind()**

|                    |                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>bind()</i> – bind a name to a socket                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre> STATUS bind (     int          s, /* socket descriptor */     struct sockaddr * name, /* name to be bound */     int          namelen /* length of name */ ) </pre>                                                                                       |
| <b>DESCRIPTION</b> | This routine associates a network address (also referred to as its "name") with a specified socket so that other processes can connect or send to it. When a socket is created with <i>socket()</i> , it belongs to an address family but has no assigned name. |

**RETURNS** OK, or ERROR if there is an invalid socket, the address is either unavailable or in use, or the socket is already bound.

**SEE ALSO** sockLib

---

## ***bindresvport()***

**NAME** *bindresvport()* – bind a socket to a privileged IP port

**SYNOPSIS**

```
STATUS bindresvport
(
 int sd, /* socket to be bound */
 struct sockaddr_in * sin /* socket address -- value/result */
)
```

**DESCRIPTION** This routine picks a port number between 600 and 1023 that is not being used by any other programs and binds the socket passed as *sd* to that port. Privileged IP ports (numbers between and including 0 and 1023) are reserved for privileged programs.

**RETURNS** OK, or ERROR if the address family specified in *sin* is not supported or the call fails.

**SEE ALSO** remLib

---

## ***binvert()***

**NAME** *binvert()* – invert the order of bytes in a buffer

**SYNOPSIS**

```
void binvert
(
 char * buf, /* pointer to buffer to invert */
 int nbytes /* number of bytes in buffer */
)
```

**DESCRIPTION** This routine inverts an entire buffer, byte by byte. For example, the buffer {1, 2, 3, 4, 5} would become {5, 4, 3, 2, 1}.

**RETURNS** N/A

**SEE ALSO** bLib

---

## ***bootBpAnchorExtract()***

- NAME** *bootBpAnchorExtract()* – extract a backplane address from a device field
- SYNOPSIS**
- ```
STATUS bootBpAnchorExtract
(
    char * string,      /* string containing adrs field */
    char * *pAnchorAdrs /* pointer where to return anchor address */
)
```
- DESCRIPTION**
- This routine extracts the optional backplane anchor address field from a boot device field. The anchor can be specified for the backplane driver by appending to the device name (i.e., "bp") an equal sign (=) and the address in hexadecimal. For example, the "boot device" field of the boot parameters could be specified as:
- ```
boot device: bp=800000
```
- In this case, the backplane anchor address would be at address 0x800000, instead of the default specified in **config.h**.
- This routine picks off the optional trailing anchor address by replacing the equal sign (=) in the specified string with an EOS and then scanning the remainder as a hex number. This number, the anchor address, is returned via the *pAnchorAdrs* pointer.
- RETURNS**
- 1 if the anchor address in *string* is specified correctly,  
0 if the anchor address in *string* is not specified, or  
-1 if an invalid anchor address is specified in *string*.
- SEE ALSO** **bootLib**

---

## ***bootChange()***

- NAME** *bootChange()* – change the boot line
- SYNOPSIS**
- ```
void bootChange (void)
```
- DESCRIPTION**
- This command changes the boot line used in the boot ROMs. This is useful during a remote login session. After changing the boot parameters, you can reboot the target with the *reboot()* command, and then terminate your login (~.) and remotely log in again. As soon as the system has rebooted, you will be logged in again.
- This command stores the new boot line in non-volatile RAM, if the target has it.

RETURNS N/A

SEE ALSO *usrLib*, *windsh*, *Tornado User's Guide: Shell*

bootLeaseExtract()

NAME *bootLeaseExtract()* – extract the lease information from an Internet address

SYNOPSIS

```
int bootLeaseExtract
(
    char *   string,      /* string containing addr field */
    u_long * pLeaseLen,   /* pointer to storage for lease duration */
    u_long * pLeaseStart /* pointer to storage for lease origin */
)
```

DESCRIPTION This routine extracts the optional lease duration and lease origin fields from an Internet address field for use with DHCP. The lease duration can be specified by appending a colon and the lease duration to the netmask field. For example, the "inet on ethernet" field of the boot parameters could be specified as:

```
inet on ethernet: 90.1.0.1:ffff0000:1000
```

If no netmask is specified, the contents of the field could be:

```
inet on ethernet: 90.1.0.1::ffffffff
```

In the first case, the lease duration for the address is 1000 seconds. The second case indicates an infinite lease, and does not specify a netmask for the address. At the beginning of the boot process, the value of the lease duration field is used to specify the requested lease duration. If the field not included, the value of `DHCP_DEFAULT_LEASE` is used instead.

The lease origin is specified with the same format as the lease duration, but is added during the boot process. The presence of the lease origin field distinguishes addresses assigned by a DHCP server from addresses entered manually. Addresses assigned by a DHCP server may be replaced if the bootstrap loader uses DHCP to obtain configuration parameters. The value of the lease origin field at the beginning of the boot process is ignored.

This routine extracts the optional lease duration by replacing the preceding colon in the specified string with an EOS and then scanning the remainder as a number. The lease duration and lease origin values are returned via the *pLeaseLen* and *pLeaseStart* pointers, if those parameters are not NULL.

bootNetmaskExtract()

- RETURNS** 2 if both lease values are specified correctly in *string*, or
 -2 if one of the two values is specified incorrectly. If only the lease duration is found, it returns:
 1 if the lease duration in *string* is specified correctly,
 0 if the lease duration is not specified in *string*, or
 -1 if an invalid lease duration is specified in *string*.
- SEE ALSO** **bootLib**

bootNetmaskExtract()

- NAME** *bootNetmaskExtract()* – extract the net mask field from an Internet address
- SYNOPSIS**
- ```
STATUS bootNetmaskExtract
(
 char * string, /* string containing addr field */
 int * pNetmask /* pointer where to return net mask */
)
```
- DESCRIPTION**
- This routine extracts the optional subnet mask field from an Internet address field. Subnet masks can be specified for an Internet interface by appending to the Internet address a colon and the net mask in hexadecimal. For example, the "inet on ethernet" field of the boot parameters could be specified as:
- ```
inet on ethernet: 90.1.0.1:ffff0000
```
- In this case, the network portion of the address (normally just 90) is extended by the subnet mask (to 90.1). This routine extracts the optional trailing subnet mask by replacing the colon in the specified string with an EOS and then scanning the remainder as a hex number. This number, the net mask, is returned via the *pNetmask* pointer.
- This routine also handles an empty netmask field used as a placeholder for the lease duration field (see *bootLeaseExtract()*). In that case, the colon separator is replaced with an EOS and the value of netmask is set to 0.
- RETURNS** 1 if the subnet mask in *string* is specified correctly,
 0 if the subnet mask in *string* is not specified, or
 -1 if an invalid subnet mask is specified in *string*.
- SEE ALSO** **bootLib**

bootParamsPrompt()

NAME *bootParamsPrompt()* – prompt for boot line parameters

SYNOPSIS

```
void bootParamsPrompt
(
    char * string /* default boot line */
)
```

DESCRIPTION This routine displays the current value of each boot parameter and prompts the user for a new value. Typing a RETURN leaves the parameter unchanged. Typing a period (.) clears the parameter.

The parameter *string* holds the initial values. The new boot line is copied over *string*. If there are no initial values, *string* is empty on entry.

RETURNS N/A

SEE ALSO **bootLib**

bootParamsShow()

NAME *bootParamsShow()* – display boot line parameters

SYNOPSIS

```
void bootParamsShow
(
    char * paramString /* boot parameter string */
)
```

DESCRIPTION This routine displays the boot parameters in the specified boot string one parameter per line.

RETURNS N/A

SEE ALSO **bootLib**

bootpMsgSend()

| | |
|--------------------|---|
| NAME | <i>bootpMsgSend()</i> – send a BOOTP request message |
| SYNOPSIS | <pre> STATUS bootpMsgSend (char * ifName, /* network interface name */ struct in_addr * pIpDest, /* destination IP address */ int port, /* port number */ BOOTP_MSG * pBootpMsg, /* pointer to BOOTP message */ u_int timeOut /* timeout in ticks */) </pre> |
| DESCRIPTION | <p>This routine sends the BOOTP message indicated by <i>pBootpMsg</i> using the network interface specified by <i>ifName</i>. The <i>pIpDest</i> argument specifies the destination IP address. In most cases, the broadcast address (255.255.255.255) is used. However, this parameter also accepts the IP address of a particular BOOTP server. That server must reside on the same subnet as the specified network interface.</p> <p>A non-zero value for <i>port</i> specifies an alternate BOOTP server port. Otherwise, the default port (67) is used.</p> <p>This routine always sets the values of the bp_op, bp_xid, and bp_secs members in the BOOTP message structure, but it allows the caller to assign values to any of the other members. However, if the bp_hlen member is 0, the routine uses the Ethernet address of the specified network interface for the bp_chaddr member and sets bp_type to 1 and bp_hlen to 6 as required for that address.</p> <p>The <i>bootpMsgSend()</i> routine will retransmit the BOOTP message if it gets no reply. The retransmission time increases exponentially but is bounded by the number of ticks specified in the <i>timeOut</i> parameter. If no reply is received within this period, an error is returned. A value of zero specifies an infinite timeout value.</p> |
| NOTE | If bp_ciaddr is specified, the BOOTP server may assume that the client will respond to an ARP request. |
| RETURNS | OK, or ERROR. |
| ERRNO | S_bootpLib_INVALID_ARGUMENT S_bootpLib_NO_BROADCASTS S_bootpLib_TIME_OUT |
| SEE ALSO | bootpLib |

bootpParamsGet()

NAME *bootpParamsGet()* – retrieve boot parameters using BOOTP

SYNOPSIS

```
STATUS bootpParamsGet
(
    char *          ifName,      /* network interface name */
    int             port,        /* optional port number */
    u_int          timeOut,     /* timeout in ticks */
    struct bootpParams * pBootpParams /* parameters descriptor */
)
```

DESCRIPTION This routine transmits a BOOTP request message over the network interface associated with *ifName*. This interface must already be attached and initialized prior to calling this routine.

A non-zero value for *port* specifies an alternate BOOTP server port. A zero value means the default BOOTP server port (67).

timeOut specifies a timeout value in ticks. If no reply is received within this period, an error is returned. Specify zero for an infinite *timeout* value.

pBootpParams is a structure pointer to a **bootpParams** structure that you can use to indicate the parameters of interest to you. The **bootpParams** structure is defined as follows:

```
struct bootpParams
{
    struct in_addr *      clientAddr;
    struct in_addr *      bootHostAddr;
    char *                bootfile;
    char *                serverName;
    struct in_addr *      netmask;
    unsigned short *      timeOffset;
    struct in_addr_list * routers;
    struct in_addr_list * timeServers;
    struct in_addr_list * nameServers;
    struct in_addr_list * dnsServers;
    struct in_addr_list * logServers;
    struct in_addr_list * cookieServers;
    struct in_addr_list * lprServers;
    struct in_addr_list * impressServers;
    struct in_addr_list * rlpServers;
    char *                clientName;
    unsigned short *      filesize;
    char *                dumpfile;
}
```

```
char *
struct in_addr *
char *
char *
unsigned char *
unsigned char *
struct in_addr_list *
unsigned short *
unsigned char *
unsigned long *
struct ushort_list *
unsigned short *
unsigned char *
struct in_addr *
unsigned char *
unsigned char *
unsigned char *
struct in_addr *
struct in_addr_list *
unsigned char *
unsigned long *
unsigned char *
unsigned char *
unsigned long *
unsigned char *
char *
struct in_addr_list *
struct in_addr_list *
char *
struct in_addr_list *
struct in_addr_list *
unsigned char *
char *
struct in_addr_list *
struct in_addr_list *
char *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
struct in_addr_list *
};

domainName;
swapServer;
rootPath;
extoptPath;
ipForward;
nonlocalSourceRoute;
policyFilter;
maxDgramSize;
ipTTL;
mtuTimeout;
mtuTable;
interfaceMTU;
allSubnetsLocal;
broadcastAddr;
maskDiscover;
maskSupplier;
routerDiscover;
routerDiscAddr;
staticRoutes;
arpTrailers;
arpTimeout;
etherPacketType;
tcpTTL;
tcpInterval;
tcpGarbage;
nisDomain;
nisServers;
ntpServers;
vendString;
nbnServers;
nbddServers;
nbNodeType;
nbScope;
xFontServers;
xDisplayManagers;
nispDomain;
nispServers;
ipAgents;
smtpServers;
pop3Servers;
nntpServers;
wwwServers;
fingerServers;
ircServers;
stServers;
stdaServers;
```

This structure allows the retrieval of any BOOTP option specified in RFC 1533. The list of 2-byte (unsigned short) values is defined as:

```
struct ushort_list
{
    unsigned char    num;
    unsigned short * shortlist;
};
```

The IP address lists use the following similar definition:

```
struct in_addr_list
{
    unsigned char    num;
    struct in_addr * addrlist;
};
```

When these lists are present, the routine stores values retrieved from the BOOTP reply in the location indicated by the **shortlist** or **addrlist** members. The amount of space available is indicated by the **num** member. When the routine returns, the **num** member indicates the actual number of entries retrieved. In the case of **bootpParams.policyFilter.num** and **bootpParams.staticRoutes.num**, the **num** member value should be interpreted as the number of IP address pairs requested and received.

The following members of the **bootpParams** structure are also used for both input and output:

clientAddr

Contains a pointer that holds the client's Internet address. On input, if it contains a non-NULL value, it is interpreted as a pointer to an Internet address of type **struct in_addr** and passed on to the BOOTP server in the **bp_ciaddr** member of the BOOTP message structure (**BOOTP_MSG**). The server will use it as a lookup field into the BOOTP database. When a reply is received, the client's assigned Internet address is copied to the **clientAddr** member.

bootHostAddr

Contains a pointer that holds the host's IP address. On input, if it contains a non-NULL value, it is interpreted as the host where the BOOTP message is to be sent. Note that this host must be local to the *pIf* network. If NULL, the BOOTP message is sent to the local broadcast address. On return, the host's IP address is copied to the **bootHostAddr** member.

On input, if the **bootpParams.bootfile** member points to a non-empty string, the contents are passed to the BOOTP server in the **bp_file** member of the BOOTP message structure (**BOOTP_MSG**). When a reply is received, the file name retrieved from the BOOTP server is copied to the **bootpParams.bootfile** member as a NULL-terminated string.

The remaining elements in the BOOTP parameters descriptor are used to select options for retrieval from the BOOTP server. The BOOTP library attempts to retrieve the values for any options whose corresponding field pointers are non-NULL values. To obtain these

parameters, the BOOTP server must support the vendor-specific options described in RFC 1048 (or its successors) and the corresponding parameters must be specified in the BOOTP server database. Where meaningful, the values are returned in host byte order.

The BOOTP request issued during system startup attempts to retrieve a subnet mask for the boot device, in addition to the host and client addresses, and the boot file name.

RETURNS OK, or ERROR if unsuccessful.

SEE ALSO **bootpLib**, **bootLib**, RFC 1048, RFC 1533

bootStringToStruct()

NAME *bootStringToStruct()* – interpret the boot parameters from the boot line

SYNOPSIS

```
char *bootStringToStruct
(
    char *      bootString, /* boot line to be parsed */
    BOOT_PARAMS * pBootParams /* where to return parsed boot line */
)
```

DESCRIPTION This routine parses the ASCII string and returns the values into the provided parameters. For a description of the format of the boot line, see the manual entry for **bootLib**

RETURNS A pointer to the last character successfully parsed plus one (points to EOS, if OK). The entire boot line is parsed.

SEE ALSO **bootLib**

bootStructToString()

NAME *bootStructToString()* – construct a boot line

SYNOPSIS

```
STATUS bootStructToString
(
    char *      paramString, /* where to return the encoded boot line */
    BOOT_PARAMS * pBootParams /* boot line structure to be encoded */
)
```


| | |
|--------------------|--|
| DESCRIPTION | This routine encodes a boot line using the specified boot parameters. For a description of the format of the boot line, see the manual entry for bootLib . |
| RETURNS | OK. |
| SEE ALSO | bootLib |

bsearch()

NAME *bsearch()* – perform a binary search (ANSI)

SYNOPSIS

```
void * bsearch
(
    const void *          key, /* element to match */
    const void *          base0, /* initial element in array */
    size_t                nmemb, /* array to search */
    size_t                size, /* size of array element */
    int (* compar) (const void *, const void *)
                        /* comparison function */
)
```

DESCRIPTION This routine searches an array of *nmemb* objects, the initial element of which is pointed to by *base0*, for an element that matches the object pointed to by *key*. The *size* of each element of the array is specified by *size*.

The comparison function pointed to by *compar* is called with two arguments that point to the *key* object and to an array element, in that order. The function shall return an integer less than, equal to, or greater than zero if the *key* object is considered, respectively, to be less than, to match, or to be greater than the array element. The array shall consist of all the elements that compare greater than the *key* object, in that order.

INCLUDE FILES **stdlib.h**

RETURNS A pointer to a matching element of the array, or a NULL pointer if no match is found. If two elements compare as equal, which element is matched is unspecified.

SEE ALSO **ansiStdlib**

bswap()

bswap()

NAME *bswap()* – swap buffers

SYNOPSIS

```
void bswap
(
    char * buf1, /* pointer to first buffer */
    char * buf2, /* pointer to second buffer */
    int   nbytes /* number of bytes to swap */
)
```

DESCRIPTION This routine exchanges the first *nbytes* of the two specified buffers.

RETURNS N/A

SEE ALSO **bLib**

bzero()

NAME *bzero()* – zero out a buffer

SYNOPSIS

```
void bzero
(
    char * buffer, /* buffer to be zeroed */
    int   nbytes  /* number of bytes in buffer */
)
```

DESCRIPTION This routine fills the first *nbytes* characters of the specified buffer with 0.

RETURNS N/A

SEE ALSO **bLib**

bzeroDoubles()

| | |
|--------------------|---|
| NAME | <i>bzeroDoubles()</i> – zero out a buffer eight bytes at a time (SPARC) |
| SYNOPSIS | <pre>STATUS bzeroDoubles (void * buffer, /* 8-byte aligned buffer */ int nbytes /* multiple of 256 bytes */)</pre> |
| DESCRIPTION | This routine fills the first <i>nbytes</i> characters of the specified buffer with 0, eight bytes at a time. The buffer address is assumed to be 8-byte aligned. The number of bytes will be rounded up to a multiple of 256 bytes. |
| RETURNS | OK, if it runs to completion. |
| SEE ALSO | <i>bALib</i> , <i>bzero()</i> |

c()

| | |
|--------------------|---|
| NAME | <i>c()</i> – continue from a breakpoint |
| SYNOPSIS | <pre>STATUS c (int task, /* task that should proceed from breakpoint */ INSTR * addr, /* address to continue at; 0 = next instruction */ INSTR * addr1 /* address for npc; 0 = instruction next to pc */)</pre> |
| DESCRIPTION | <p>This routine continues the execution of a task that has stopped at a breakpoint. To execute, enter:</p> <pre>-> c [task [,addr[,addr1]]]</pre> <p>If <i>task</i> is omitted or zero, the last task referenced is assumed. If <i>addr</i> is non-zero, the program counter is changed to <i>addr</i>; if <i>addr1</i> is non-zero, the next program counter is changed to <i>addr1</i>, and the task is continued.</p> |
| CAVEAT | When a task is continued, <i>c()</i> does not distinguish between a suspended task or a task suspended by the debugger. Therefore, its use should be restricted to only those tasks being debugged. |

cacheArchClearEntry()

- NOTE** The next program counter, *addr1*, is currently supported only by SPARC.
- RETURNS** OK, or ERROR if the specified task does not exist.
- SEE ALSO** **dbgLib**, **tr()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

cacheArchClearEntry()

NAME *cacheArchClearEntry()* – clear an entry from a cache (68K, x86)

SYNOPSIS

```
STATUS cacheArchClearEntry
(
    CACHE_TYPE cache, /* cache to clear entry for */
    void *      address /* entry to clear */
)
```

DESCRIPTION This routine clears a specified entry from the specified cache.

For 68040 processors, this routine clears the cache line from the cache in which the cache entry resides.

For the MC68060 processor, when the instruction cache is cleared (invalidated) the branch cache is also invalidated by the hardware. One line in the branch cache cannot be invalidated so each time the branch cache is entirely invalidated.

For 386 processors and PENTIUMPRO processors with SNOOP_ENABLED data cache mode, this routine does nothing.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO **cacheArchLib**

cacheArchLibInit()

NAME *cacheArchLibInit()* – initialize the cache library

SYNOPSIS

```
STATUS cacheArchLibInit
(
    CACHE_MODE instMode, /* instruction cache mode */
    CACHE_MODE dataMode /* data cache mode */
)
```

DESCRIPTION This routine initializes the cache library for the following processor cache families: Motorola 68K, Intel 960, Intel x86, PowerPC ARM, and the Solaris, HP-UX, and NT simulators. It initializes the function pointers and configures the caches to the specified cache modes.

68K PROCESSORS The caching modes vary for members of the 68K processor family:

| | | |
|--------------|---|--------------------------|
| 68020 | CACHE_WRITETHROUGH | (instruction cache only) |
| 68030 | CACHE_WRITETHROUGH CACHE_BURST_ENABLE CACHE_BURST_DISABLE CACHE_WRITEALLOCATE | (data cache only) |
| | CACHE_NO_WRITEALLOCATE | (data cache only) |
| 68040 | CACHE_WRITETHROUGH CACHE_COPYBACK CACHE_INH_SERIAL CACHE_INH_NONSERIAL CACHE_BURST_ENABLE CACHE_NO_WRITEALLOCATE | (data cache only) |
| 68060 | CACHE_WRITETHROUGH CACHE_COPYBACK CACHE_INH_PRECISE CACHE_INH_IMPRECISE CACHE_BURST_ENABLE | (data cache only) |

The write-through, copy-back, serial, non-serial, precise and non precise modes change the state of the data transparent translation register (DTTR0) CM bits. Only DTTR0 is modified, since it typically maps DRAM space.

X86 PROCESSORS The caching mode **CACHE_WRITETHROUGH** is available for the x86 processor family.

POWER PC PROCESSORS

Modes should be set before caching is enabled. If two contradictory flags are set (for example, enable/disable), no action is taken for any of the input flags.

ARM PROCESSORS The caching capabilities and modes vary for members of the ARM processor family. All caches are provided on-chip, so cache support is mostly an architecture issue, not a BSP issue. However, the memory map is BSP-specific and some functions need knowledge of the memory map, so they have to be provided in the BSP.

ARM7TDMI (In ARM or Thumb state)

No cache or MMU at all. Dummy routine provided, so that **INCLUDE_CACHE_SUPPORT** can be defined (the default BSP configuration).

ARM710A

Combined instruction and data cache. Actually a write-through cache, but separate

cacheClear()

write-buffer effectively makes this a copy-back cache if the write-buffer is enabled. Use write-through/copy-back argument to decide whether to enable write buffer. Data and instruction cache modes must be identical.

ARM810

Combined instruction and data cache. Write-through and copy-back cache modes, but separate write-buffer effectively makes even write-through a copy-back cache as all writes are buffered, when cache is enabled. Data and instruction cache modes must be identical.

ARMSA110

Separate instruction and data caches. Write-through and copy-back cache mode for data, but separate write-buffer effectively makes even write-through a copy-back cache as all writes are buffered, when cache is enabled.

RETURNS OK

SEE ALSO cacheArchLib

cacheClear()

NAME cacheClear() – clear all or some entries from a cache

SYNOPSIS

```
STATUS cacheClear
(
    CACHE_TYPE cache, /* cache to clear */
    void *      address, /* virtual address */
    size_t     bytes /* number of bytes to clear */
)
```

DESCRIPTION This routine flushes and invalidates all or some entries in the specified cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO cacheLib

cacheCy604ClearLine()

NAME *cacheCy604ClearLine()* – clear a line from a CY7C604 cache

SYNOPSIS

```
STATUS cacheCy604ClearLine
(
    CACHE_TYPE cache, /* cache to clear */
    void *      address /* virtual address */
)
```

DESCRIPTION This routine flushes and invalidates a specified line from the specified CY7C604 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO *cacheCy604Lib*

cacheCy604ClearPage()

NAME *cacheCy604ClearPage()* – clear a page from a CY7C604 cache

SYNOPSIS

```
STATUS cacheCy604ClearPage
(
    CACHE_TYPE cache, /* cache to clear */
    void *      address /* virtual address */
)
```

DESCRIPTION This routine flushes and invalidates the specified page from the specified CY7C604 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO *cacheCy604Lib*

cacheCy604ClearRegion()

NAME *cacheCy604ClearRegion()* – clear a region from a CY7C604 cache

SYNOPSIS `STATUS cacheCy604ClearRegion`
(
 CACHE_TYPE cache, /* cache to clear */
 void * address /* virtual address */
)

DESCRIPTION This routine flushes and invalidates a specified region from the specified CY7C604 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO *cacheCy604Lib*

cacheCy604ClearSegment()

NAME *cacheCy604ClearSegment()* – clear a segment from a CY7C604 cache

SYNOPSIS `STATUS cacheCy604ClearSegment`
(
 CACHE_TYPE cache, /* cache to clear */
 void * address /* virtual address */
)

DESCRIPTION This routine flushes and invalidates a specified segment from the specified CY7C604 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO *cacheCy604Lib*

cacheCy604LibInit()

| | |
|--------------------|--|
| NAME | <i>cacheCy604LibInit()</i> – initialize the Cypress CY7C604 cache library |
| SYNOPSIS | <pre>STATUS cacheCy604LibInit (CACHE_MODE instMode, /* instruction cache mode */ CACHE_MODE dataMode /* data cache mode */)</pre> |
| DESCRIPTION | <p>This routine initializes the function pointers for the Cypress CY7C604 cache library. The board support package can select this cache library by assigning the function pointer <code>sysCacheLibInit</code> to <i>cacheCy604LibInit()</i>.</p> <p>The available cache modes are <code>CACHE_WRITETHROUGH</code> and <code>CACHE_COPYBACK</code>. Write-through uses "no-write allocate"; copyback uses "write allocate."</p> |
| RETURNS | OK, or ERROR if cache control is not supported. |
| SEE ALSO | <code>cacheCy604Lib</code> |

cacheDisable()

| | |
|--------------------|--|
| NAME | <i>cacheDisable()</i> – disable the specified cache |
| SYNOPSIS | <pre>STATUS cacheDisable (CACHE_TYPE cache /* cache to disable */)</pre> |
| DESCRIPTION | This routine flushes the cache and disables the instruction or data cache. |
| RETURNS | OK, or ERROR if the cache type is invalid or the cache control is not supported. |
| SEE ALSO | <code>cacheLib</code> |

cacheDmaFree()

NAME *cacheDmaFree()* – free the buffer acquired with *cacheDmaMalloc()*

SYNOPSIS

```
STATUS cacheDmaFree
(
    void * pBuf /* pointer to malloc/free buffer */
)
```

DESCRIPTION This routine frees the buffer returned by *cacheDmaMalloc()*.

RETURNS OK, or ERROR if the cache control is not supported.

SEE ALSO *cacheLib*

cacheDmaMalloc()

NAME *cacheDmaMalloc()* – allocate a cache-safe buffer for DMA devices and drivers

SYNOPSIS

```
void * cacheDmaMalloc
(
    size_t bytes /* number of bytes to allocate */
)
```

DESCRIPTION This routine returns a pointer to a section of memory that will not experience any cache coherency problems. Function pointers in the `CACHE_FUNCS` structure provide access to DMA support routines.

RETURNS A pointer to the cache-safe buffer, or NULL.

SEE ALSO *cacheLib*

cacheDrvFlush()

NAME *cacheDrvFlush()* – flush the data cache for drivers

SYNOPSIS

```
STATUS cacheDrvFlush
(
    CACHE_FUNCS * pFuncs, /* pointer to CACHE_FUNCS */
    void *      address, /* virtual address */
    size_t      bytes    /* number of bytes to flush */
)
```

DESCRIPTION This routine flushes the data cache entries using the function pointer from the specified set.

RETURNS OK, or ERROR if the cache control is not supported.

SEE ALSO *cacheLib*

cacheDrvInvalidate()

NAME *cacheDrvInvalidate()* – invalidate data cache for drivers

SYNOPSIS

```
STATUS cacheDrvInvalidate
(
    CACHE_FUNCS * pFuncs, /* pointer to CACHE_FUNCS */
    void *      address, /* virtual address */
    size_t      bytes    /* no. of bytes to invalidate */
)
```

DESCRIPTION This routine invalidates the data cache entries using the function pointer from the specified set.

RETURNS OK, or ERROR if the cache control is not supported.

SEE ALSO *cacheLib*

cacheDrvPhysToVirt()

NAME *cacheDrvPhysToVirt()* – translate a physical address for drivers

SYNOPSIS

```
void * cacheDrvPhysToVirt
(
    CACHE_FUNCS * pFuncs, /* pointer to CACHE_FUNCS */
    void *      address /* physical address */
)
```

DESCRIPTION This routine performs a physical-to-virtual address translation using the function pointer from the specified set.

RETURNS The virtual address that maps to the physical address argument.

SEE ALSO *cacheLib*

cacheDrvVirtToPhys()

NAME *cacheDrvVirtToPhys()* – translate a virtual address for drivers

SYNOPSIS

```
void * cacheDrvVirtToPhys
(
    CACHE_FUNCS * pFuncs, /* pointer to CACHE_FUNCS */
    void *      address /* virtual address */
)
```

DESCRIPTION This routine performs a virtual-to-physical address translation using the function pointer from the specified set.

RETURNS The physical address translation of a virtual address argument.

SEE ALSO *cacheLib*

cacheEnable()

| | |
|--------------------|--|
| NAME | <i>cacheEnable()</i> – enable the specified cache |
| SYNOPSIS | <pre>STATUS cacheEnable (CACHE_TYPE cache /* cache to enable */)</pre> |
| DESCRIPTION | This routine invalidates the cache tags and enables the instruction or data cache. |
| RETURNS | OK, or ERROR if the cache type is invalid or the cache control is not supported. |
| SEE ALSO | cacheLib |

cacheFlush()

| | |
|--------------------|---|
| NAME | <i>cacheFlush()</i> – flush all or some of a specified cache |
| SYNOPSIS | <pre>STATUS cacheFlush (CACHE_TYPE cache, /* cache to flush */ void * address, /* virtual address */ size_t bytes /* number of bytes to flush */)</pre> |
| DESCRIPTION | This routine flushes (writes to memory) all or some of the entries in the specified cache. Depending on the cache design, this operation may also invalidate the cache tags. For write-through caches, no work needs to be done since RAM already matches the cached entries. Note that write buffers on the chip may need to be flushed to complete the flush. |
| RETURNS | OK, or ERROR if the cache type is invalid or the cache control is not supported. |
| SEE ALSO | cacheLib |

cacheI960CxIC1kLoadNLock()

| | |
|--------------------|---|
| NAME | <i>cacheI960CxIC1kLoadNLock()</i> – load and lock I960Cx 1KB instruction cache (i960) |
| SYNOPSIS | <pre>void cacheI960CxIC1kLoadNLock (void * address)</pre> |
| DESCRIPTION | This routine loads and locks the I960Cx 1KB instruction cache. The loaded address must be an address of a quad-word aligned block of memory. The instructions loaded into the cache can only be accessed by selected interrupts which vector to the addresses of these instructions. The load-and-lock mechanism selectively optimizes latency and throughput for interrupts. |
| RETURNS | N/A |
| SEE ALSO | <i>cacheI960CxALib</i> |

cacheI960CxICDisable()

| | |
|--------------------|---|
| NAME | <i>cacheI960CxICDisable()</i> – disable the I960Cx instruction cache (i960) |
| SYNOPSIS | <pre>void cacheI960CxICDisable (void)</pre> |
| DESCRIPTION | This routine disables the I960Cx instruction cache. |
| RETURNS | N/A |
| SEE ALSO | <i>cacheI960CxALib</i> |

cacheI960CxICEnable()

| | |
|-----------------|---|
| NAME | <i>cacheI960CxICEnable()</i> – enable the I960Cx instruction cache (i960) |
| SYNOPSIS | <pre>void cacheI960CxICEnable (void)</pre> |

DESCRIPTION This routine enables the I960Cx instruction cache.

RETURNS N/A

SEE ALSO `cacheI960CxALib`

cacheI960CxICInvalidate()

NAME `cacheI960CxICInvalidate()` – invalidate the I960Cx instruction cache (i960)

SYNOPSIS `void cacheI960CxICInvalidate (void)`

DESCRIPTION

SEE ALSO `cacheI960CxALib`

cacheI960CxICLoadNLock()

NAME `cacheI960CxICLoadNLock()` – load and lock I960Cx 512-byte instruction cache (i960)

SYNOPSIS `void cacheI960CxICLoadNLock
(
void * address
)`

DESCRIPTION This routine loads and locks the I960Cx 512-byte instruction cache. The loaded address must be an address of a quad-word aligned block of memory. The instructions loaded into the cache can only be accessed by selected interrupts which vector to the addresses of these instructions. The load-and-lock mechanism selectively optimizes latency and throughput for interrupts.

RETURNS N/A

SEE ALSO `cacheI960CxALib`

cacheI960CxLibInit()

NAME *cacheI960CxLibInit()* – initialize the I960Cx cache library (i960)

SYNOPSIS

```
STATUS cacheI960CxLibInit
(
    CACHE_MODE instMode, /* instruction cache mode */
    CACHE_MODE dataMode /* data cache mode */
)
```

DESCRIPTION This routine initializes the function pointers for the I960Cx cache library. The board support package can select this cache library by calling this routine.

RETURNS OK.

SEE ALSO *cacheI960CxLib*

cacheI960JxDCCoherent()

NAME *cacheI960JxDCCoherent()* – ensure data cache coherency (i960)

SYNOPSIS

```
void cacheI960JxDCCoherent ( void )
```

DESCRIPTION This routine ensures coherency by invalidating data cache on the I960Jx.

RETURNS N/A

SEE ALSO *cacheI960JxA Lib*

cacheI960JxDCDisable()

NAME *cacheI960JxDCDisable()* – disable the I960Jx data cache (i960)

SYNOPSIS

```
void cacheI960JxDCDisable ( void )
```

DESCRIPTION This routine disables the I960Jx data cache.

RETURNS N/A

SEE ALSO `cacheI960JxALib`

cacheI960JxDCEnable()

NAME *cacheI960JxDCEnable()* – enable the I960Jx data cache (i960)

SYNOPSIS `void cacheI960JxDCEnable (void)`

DESCRIPTION This routine enables the I960Jx data cache.

RETURNS N/A

SEE ALSO `cacheI960JxALib`

cacheI960JxDCFlush()

NAME *cacheI960JxDCFlush()* – flush the I960Jx data cache (i960)

SYNOPSIS `void cacheI960JxDCFlush ()`

DESCRIPTION This routine flushes the I960Jx data cache.

RETURNS N/A

SEE ALSO `cacheI960JxALib`

cacheI960JxDCInvalidate()

NAME *cacheI960JxDCInvalidate()* – invalidate the I960Jx data cache (i960)

SYNOPSIS `void cacheI960JxDCInvalidate (void)`

DESCRIPTION This routine invalidates the I960Jx data cache.

RETURNS N/A

SEE ALSO cacheI960JxALib

cacheI960JxDCStatusGet()

NAME *cacheI960JxDCStatusGet()* – get the I960Jx data cache status (i960)

SYNOPSIS

```
void cacheI960JxDCStatusGet  
(  
)
```

DESCRIPTION This routine gets the I960Jx data cache status.

RETURNS N/A

SEE ALSO cacheI960JxALib

cacheI960JxICDisable()

NAME *cacheI960JxICDisable()* – disable the I960Jx instruction cache (i960)

SYNOPSIS

```
void cacheI960JxICDisable (void)
```

DESCRIPTION This routine disables the I960Jx instruction cache.

RETURNS N/A

SEE ALSO cacheI960JxALib

cacheI960JxICEnable()

NAME *cacheI960JxICEnable()* – enable the I960Jx instruction cache (i960)

SYNOPSIS

```
void cacheI960JxICEnable ( void )
```

DESCRIPTION This routine enables the I960Jx instruction cache.

RETURNS N/A

SEE ALSO *cacheI960JxALib*

cacheI960JxICFlush()

NAME *cacheI960JxICFlush()* – flush the I960Jx instruction cache (i960)

SYNOPSIS `void cacheI960JxICFlush ()`

DESCRIPTION This routine flushes the I960Jx instruction cache.

RETURNS N/A

SEE ALSO *cacheI960JxALib*

cacheI960JxICInvalidate()

NAME *cacheI960JxICInvalidate()* – invalidate the I960Jx instruction cache (i960)

SYNOPSIS `void cacheI960JxICInvalidate (void)`

SEE ALSO *cacheI960JxALib*

cacheI960JxICLoadNLock()

NAME *cacheI960JxICLoadNLock()* – load and lock the I960Jx instruction cache (i960)

SYNOPSIS `void cacheI960JxICLoadNLock ()`

DESCRIPTION This routine loads and locks the I960Jx instruction cache.

RETURNS N/A

SEE ALSO *cacheI960JxALib*

cacheI960JxICLockingStatusGet()

NAME *cacheI960JxICLockingStatusGet()* – get the I960Jx I-cache locking status (i960)

SYNOPSIS

```
void cacheI960JxICLockingStatusGet  
(  
)
```

DESCRIPTION This routine gets the I960Jx instruction cache locking status.

RETURNS N/A

SEE ALSO *cacheI960JxA Lib*

cacheI960JxICStatusGet()

NAME *cacheI960JxICStatusGet()* – get the I960Jx instruction cache status (i960)

SYNOPSIS

```
void cacheI960JxICStatusGet  
(  
)
```

DESCRIPTION This routine gets the I960Jx instruction cache status.

RETURNS N/A

SEE ALSO *cacheI960JxA Lib*

cacheI960JxLibInit()

NAME *cacheI960JxLibInit()* – initialize the I960Jx cache library (i960)

SYNOPSIS

```
STATUS cacheI960JxLibInit  
(  
    CACHE_MODE instMode, /* instruction cache mode */  
    CACHE_MODE dataMode /* data cache mode */  
)
```

| | |
|--------------------|---|
| DESCRIPTION | This routine initializes the function pointers for the I960Jx cache library. The board support package can select this cache library by calling this routine. |
| RETURNS | OK. |
| SEE ALSO | cacheI960JxLib |

cacheInvalidate()

| | |
|--------------------|--|
| NAME | <i>cacheInvalidate()</i> – invalidate all or some of a specified cache |
| SYNOPSIS | <pre>STATUS cacheInvalidate (CACHE_TYPE cache, /* cache to invalidate */ void * address, /* virtual address */ size_t bytes /* number of bytes to invalidate */) </pre> |
| DESCRIPTION | This routine invalidates all or some of the entries in a cache. Depending on cache design, the invalidation may be similar to the flush, or the tags may be invalidated directly. |
| RETURNS | OK, or ERROR if the cache type is invalid or the cache control is not supported. |
| SEE ALSO | cacheLib |

cacheLibInit()

| | |
|--------------------|---|
| NAME | <i>cacheLibInit()</i> – initialize the cache library for a processor architecture |
| SYNOPSIS | <pre>STATUS cacheLibInit (CACHE_MODE instMode, /* inst cache mode */ CACHE_MODE dataMode /* data cache mode */) </pre> |
| DESCRIPTION | This routine initializes the function pointers for the appropriate cache library. For architectures with more than one cache implementation, the board support package must select the appropriate cache library with sysCacheLibInit . Systems without cache coherency problems (i.e., bus snooping) should NULLify the flush and invalidate function |

cacheLock()

pointers in the **cacheLib** structure to enhance driver and overall system performance. This can be done in *sysHwInit()*.

RETURNS OK, or ERROR if there is no cache library installed.

SEE ALSO **cacheLib**

cacheLock()

NAME *cacheLock()* – lock all or part of a specified cache

SYNOPSIS

```
STATUS cacheLock
(
    CACHE_TYPE cache, /* cache to lock */
    void *      address, /* virtual address */
    size_t     bytes /* number of bytes to lock */
)
```

DESCRIPTION This routine locks all (global) or some (local) entries in the specified cache. Cache locking is useful in real-time systems. Not all caches can perform locking.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO **cacheLib**

cacheMb930ClearLine()

NAME *cacheMb930ClearLine()* – clear a line from an MB86930 cache

SYNOPSIS

```
STATUS cacheMb930ClearLine
(
    CACHE_TYPE cache, /* cache to clear entry */
    void *      address /* virtual address */
)
```

DESCRIPTION This routine flushes and invalidates a specified line from the specified MB86930 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO **cacheMb930Lib**

cacheMb930LibInit()

NAME *cacheMb930LibInit()* – initialize the Fujitsu MB86930 cache library

SYNOPSIS

```
STATUS cacheMb930LibInit
(
    CACHE_MODE instMode, /* instruction cache mode */
    CACHE_MODE dataMode /* data cache mode */
)
```

DESCRIPTION This routine installs the function pointers for the Fujitsu MB86930 cache library and performs other necessary cache library initialization. The board support package selects this cache library by setting the function pointer **sysCacheLibInit** equal to *cacheMb930LibInit()*. Note that **sysCacheLibInit** must be initialized on declaration, placing it in the ".data" section.

This routine invalidates the cache tags and leaves the cache disabled. It should only be called during initialization, before any cache locking has taken place.

The only available mode for the MB86930 is **CACHE_WRITETHROUGH**.

RETURNS OK, or ERROR if cache control is not supported.

SEE ALSO *cacheMb930Lib*

cacheMb930LockAuto()

NAME *cacheMb930LockAuto()* – enable MB86930 automatic locking of kernel instructions/data

SYNOPSIS

```
void cacheMb930LockAuto (void)
```

DESCRIPTION This routine enables automatic cache locking of kernel instructions and data into MB86930 caches. Once entries are locked into the caches, they cannot be unlocked.

RETURNS N/A

SEE ALSO *cacheMb930Lib*

cacheMicroSparcLibInit()

| | |
|--------------------|---|
| NAME | <i>cacheMicroSparcLibInit()</i> – initialize the microSPARC cache library |
| SYNOPSIS | <pre>STATUS cacheMicroSparcLibInit (CACHE_MODE instMode, /* instruction cache mode */ CACHE_MODE dataMode /* data cache mode */)</pre> |
| DESCRIPTION | <p>This routine initializes the function pointers for the microSPARC cache library. The board support package can select this cache library by assigning the function pointer sysCacheLibInit to <i>cacheMicroSparcLibInit()</i>.</p> <p>The only available cache mode is CACHE_WRITETHROUGH.</p> |
| RETURNS | OK, or ERROR if cache control is not supported. |
| SEE ALSO | cacheMicroSparcLib |

cachePipeFlush()

| | |
|--------------------|--|
| NAME | <i>cachePipeFlush()</i> – flush processor write buffers to memory |
| SYNOPSIS | <pre>STATUS cachePipeFlush (void)</pre> |
| DESCRIPTION | <p>This routine forces the processor output buffers to write their contents to RAM. A cache flush may have forced its data into the write buffers, then the buffers need to be flushed to RAM to maintain coherency.</p> |
| RETURNS | OK, or ERROR if the cache control is not supported. |
| SEE ALSO | cacheLib |

cacheR3kDsize()

NAME *cacheR3kDsize()* – return the size of the R3000 data cache

SYNOPSIS `ULONG cacheR3kDsize (void)`

DESCRIPTION This routine returns the size of the R3000 data cache. Generally, this value should be placed into the value *cacheDCacheSize* for use by other routines.

RETURNS The size of the data cache in bytes.

SEE ALSO *cacheR3kALib*

cacheR3kIsize()

NAME *cacheR3kIsize()* – return the size of the R3000 instruction cache

SYNOPSIS `ULONG cacheR3kIsize (void)`

DESCRIPTION This routine returns the size of the R3000 instruction cache. Generally, this value should be placed into the value *cacheDCacheSize* for use by other routines.

RETURNS The size of the instruction cache in bytes.

SEE ALSO *cacheR3kALib*

cacheR3kLibInit()

NAME *cacheR3kLibInit()* – initialize the R3000 cache library

SYNOPSIS `STATUS cacheR3kLibInit
(
 CACHE_MODE instMode, /* instruction cache mode */
 CACHE_MODE dataMode /* data cache mode */
)`

DESCRIPTION This routine initializes the function pointers for the R3000 cache library. The board support package can select this cache library by calling this routine.

cacheR4kLibInit()

RETURNS OK.

SEE ALSO cacheR3kLib

cacheR4kLibInit()

NAME *cacheR4kLibInit()* – initialize the R4000 cache library

SYNOPSIS

```
STATUS cacheR4kLibInit
(
    CACHE_MODE instMode, /* instruction cache mode */
    CACHE_MODE dataMode /* data cache mode */
)
```

DESCRIPTION This routine initializes the function pointers for the R4000 cache library. The board support package can select this cache library by assigning the function pointer *sysCacheLibInit* to *cacheR4kLibInit()*.

RETURNS OK.

SEE ALSO cacheR4kLib

cacheR33kLibInit()

NAME *cacheR33kLibInit()* – initialize the R33000 cache library

SYNOPSIS

```
STATUS cacheR33kLibInit
(
    CACHE_MODE instMode, /* instruction cache mode */
    CACHE_MODE dataMode /* data cache mode */
)
```

DESCRIPTION This routine initializes the function pointers for the R33000 cache library. The board support package can select this cache library by calling this routine.

RETURNS OK.

SEE ALSO cacheR33kLib

cacheR333x0LibInit()

NAME *cacheR333x0LibInit()* – initialize the R333x0 cache library

SYNOPSIS

```
STATUS cacheR333x0LibInit
(
    CACHE_MODE instMode, /* instruction cache mode */
    CACHE_MODE dataMode /* data cache mode */
)
```

DESCRIPTION This routine initializes the function pointers for the R333x0 cache library. The board support package can select this cache library by calling this routine.

RETURNS OK.

SEE ALSO *cacheR333x0Lib*

cacheStoreBufDisable()

NAME *cacheStoreBufDisable()* – disable the store buffer (MC68060 only)

SYNOPSIS

```
void cacheStoreBufDisable (void)
```

DESCRIPTION This routine resets the ESB bit of the Cache Control Register (CACR) to disable the store buffer.

RETURNS N/A

SEE ALSO *cacheArchLib*

cacheStoreBufEnable()

NAME *cacheStoreBufEnable()* – enable the store buffer (MC68060 only)

SYNOPSIS

```
void cacheStoreBufEnable (void)
```

cacheSun4ClearContext()

DESCRIPTION This routine sets the ESB bit of the Cache Control Register (CACR) to enable the store buffer. To maximize performance, the four-entry first-in-first-out (FIFO) store buffer is used to defer pending writes to writethrough or cache-inhibited imprecise pages.

RETURNS N/A

SEE ALSO **cacheArchLib**

cacheSun4ClearContext()

NAME *cacheSun4ClearContext()* – clear a specific context from a Sun-4 cache

SYNOPSIS

```
STATUS cacheSun4ClearContext
(
    CACHE_TYPE cache, /* cache to clear */
    void *      address /* virtual address */
)
```

DESCRIPTION This routine flushes and invalidates a specified context from the specified Sun-4 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO **cacheSun4Lib**

cacheSun4ClearLine()

NAME *cacheSun4ClearLine()* – clear a line from a Sun-4 cache

SYNOPSIS

```
STATUS cacheSun4ClearLine
(
    CACHE_TYPE cache, /* cache to clear */
    void *      address /* virtual address */
)
```

DESCRIPTION This routine flushes and invalidates a specified line from the specified Sun-4 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO **cacheSun4Lib**

cacheSun4ClearPage()

NAME *cacheSun4ClearPage()* – clear a page from a Sun-4 cache

SYNOPSIS

```
STATUS cacheSun4ClearPage
(
    CACHE_TYPE cache, /* cache to clear */
    void *      address /* virtual address */
)
```

DESCRIPTION This routine flushes and invalidates a specified page from the specified Sun-4 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO *cacheSun4Lib*

cacheSun4ClearSegment()

NAME *cacheSun4ClearSegment()* – clear a segment from a Sun-4 cache

SYNOPSIS

```
STATUS cacheSun4ClearSegment
(
    CACHE_TYPE cache, /* cache to clear */
    void *      address /* virtual address */
)
```

DESCRIPTION This routine flushes and invalidates a specified segment from the specified Sun-4 cache.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO *cacheSun4Lib*

cacheSun4LibInit()

| | |
|--------------------|---|
| NAME | <i>cacheSun4LibInit()</i> – initialize the Sun-4 cache library |
| SYNOPSIS | <pre>STATUS cacheSun4LibInit (CACHE_MODE instMode, /* instruction cache mode */ CACHE_MODE dataMode /* data cache mode */)</pre> |
| DESCRIPTION | <p>This routine initializes the function pointers for the Sun Microsystems Sun-4 cache library. The board support package can select this cache library by assigning the function pointer <code>sysCacheLibInit</code> to <i>cacheSun4LibInit()</i>.</p> <p>The only available mode for the Sun-4 cache is <code>CACHE_WRITETHROUGH</code>.</p> |
| RETURNS | OK, or ERROR if cache control is not supported. |
| SEE ALSO | <code>cacheSun4Lib</code> |

cacheTextUpdate()

| | |
|--------------------|---|
| NAME | <i>cacheTextUpdate()</i> – synchronize the instruction and data caches |
| SYNOPSIS | <pre>STATUS cacheTextUpdate (void * address, /* virtual address */ size_t bytes /* number of bytes to sync */)</pre> |
| DESCRIPTION | <p>This routine flushes the data cache, then invalidates the instruction cache. This operation forces the instruction cache to fetch code that may have been created via the data path.</p> |
| RETURNS | OK, or ERROR if the cache control is not supported. |
| SEE ALSO | <code>cacheLib</code> |

cacheTiTms390LibInit()

NAME *cacheTiTms390LibInit()* – initialize the TI TMS390 cache library

SYNOPSIS

```
STATUS cacheTiTms390LibInit
(
    CACHE_MODE instMode, /* instruction cache mode */
    CACHE_MODE dataMode /* data cache mode */
)
```

DESCRIPTION This routine initializes the function pointers for the TI TMS390 cache library. The board support package can select this cache library by assigning the function pointer **sysCacheLibInit** to *cacheTiTms390LibInit()*.

The only available cache mode is **CACHE_COPYBACK**.

RETURNS OK, or ERROR if cache control is not supported.

SEE ALSO **cacheTiTms390Lib**

cacheTiTms390PhysToVirt()

NAME *cacheTiTms390PhysToVirt()* – translate a physical address for drivers

SYNOPSIS

```
void * cacheTiTms390PhysToVirt
(
    void * address /* physical address */
)
```

DESCRIPTION This routine performs a 32-bit physical to 32-bit virtual address translation in the current context.

It works for only DRAM addresses of the first EMC.

It guesses likely virtual addresses, and checks its guesses with **VM_TRANSLATE**. A likely virtual address is the same as the physical address, or some multiple of 16M less. If any match, it succeeds. If all guesses are wrong, it fails.

RETURNS The virtual address that maps to the physical address bits [31:0] argument, or NULL if it fails.

RETURNS N/A

SEE ALSO `cacheTiTms390Lib`

cacheTiTms390VirtToPhys()

NAME `cacheTiTms390VirtToPhys()` – translate a virtual address for cacheLib

SYNOPSIS

```
void * cacheTiTms390VirtToPhys
(
    void * address /* virtual address */
)
```

DESCRIPTION This routine performs a 32-bit virtual to 32-bit physical address translation in the current context.

RETURNS The physical address translation bits [31:0] of a virtual address argument, or NULL if the virtual address is not valid, or the physical address does not fit in 32 bits.

RETURNS N/A

SEE ALSO `cacheTiTms390Lib`

cacheUnlock()

NAME `cacheUnlock()` – unlock all or part of a specified cache

SYNOPSIS

```
STATUS cacheUnlock
(
    CACHE_TYPE cache, /* cache to unlock */
    void * address, /* virtual address */
    size_t bytes /* number of bytes to unlock */
)
```

DESCRIPTION This routine unlocks all (global) or some (local) entries in the specified cache. Not all caches can perform unlocking.

RETURNS OK, or ERROR if the cache type is invalid or the cache control is not supported.

SEE ALSO `cacheLib`

calloc()

| | |
|--------------------|---|
| NAME | <i>calloc()</i> – allocate space for an array (ANSI) |
| SYNOPSIS | <pre>void *calloc (size_t elemNum, /* number of elements */ size_t elemSize /* size of elements */)</pre> |
| DESCRIPTION | This routine allocates a block of memory for an array that contains <i>elemNum</i> elements of size <i>elemSize</i> . This space is initialized to zeros. |
| RETURNS | A pointer to the block, or NULL if the call fails. |
| SEE ALSO | memLib , <i>American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: General Utilities (stdlib.h)</i> |

cbrt()

| | |
|----------------------|---|
| NAME | <i>cbrt()</i> – compute a cube root |
| SYNOPSIS | <pre>double cbrt (double x /* value to compute the cube root of */)</pre> |
| DESCRIPTION | This routine returns the cube root of <i>x</i> in double precision. |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision cube root of <i>x</i> . |
| SEE ALSO | mathALib |

cbrtf()

cbrtf()

| | |
|----------------------|---|
| NAME | <i>cbrtf()</i> – compute a cube root |
| SYNOPSIS | <pre>float cbrtf (float x /* argument */)</pre> |
| DESCRIPTION | This routine returns the cube root of <i>x</i> in single precision. |
| INCLUDE FILES | math.h |
| RETURNS | The single-precision cube root of <i>x</i> . |
| SEE ALSO | mathALib |

cd()

| | |
|--------------------|--|
| NAME | <i>cd()</i> – change the default directory |
| SYNOPSIS | <pre>STATUS cd (char * name /* new directory name */)</pre> |
| DESCRIPTION | <p>This command sets the default directory to <i>name</i>. The default directory is a device name, optionally followed by a directory local to that device.</p> <p>To change to a different directory, specify one of the following:</p> <ul style="list-style-type: none"> – an entire path name with a device name, possibly followed by a directory name. The entire path name will be changed. – a directory name starting with a ~ or / or \$. The directory part of the path, immediately after the device name, will be replaced with the new directory name. – a directory name to be appended to the current default directory. The directory name will be appended to the current default directory. <p>An instance of ".." indicates one level up in the directory tree.</p> |

Note that when accessing a remote file system via RSH or FTP, the VxWorks network device must already have been created using *netDevCreate()*.

WARNING The *cd()* command does little checking that *name* represents a valid path. If the path is invalid *cd()* may return OK, but subsequent calls that depend on the default path will fail.

EXAMPLES The following example changes the directory to device */fd0/*:

```
-> cd "/fd0/"
```

This example changes the directory to device **wrs:** with the local directory *~leslie/target*:

```
-> cd "wrs:~leslie/target"
```

After the previous command, the following changes the directory to **wrs:~leslie/target/config**:

```
-> cd "config"
```

After the previous command, the following changes the directory to **wrs:~leslie/target/demo**:

```
-> cd "../demo"
```

After the previous command, the following changes the directory to **wrs:/etc**.

```
-> cd "/etc"
```

Note that *~* can be used only on network devices (RSH or FTP).

RETURNS OK or ERROR.

SEE ALSO *usrLib*, *pwd()*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

cd2400HrdInit()

NAME *cd2400HrdInit()* – initialize the chip

SYNOPSIS

```
void cd2400HrdInit  
(  
    CD2400_QUSART * pQusart /* chip to reset */  
)
```

DESCRIPTION This routine initializes the chip and the four channels.

SEE ALSO *cd2400Sio*

cd2400Int()

NAME *cd2400Int()* – handle special status interrupts

SYNOPSIS

```
void cd2400Int
(
    CD2400_CHAN * pChan
)
```

DESCRIPTION This routine handles special status interrupts from the MPCC.

SEE ALSO *cd2400Sio*

cd2400IntRx()

NAME *cd2400IntRx()* – handle receiver interrupts

SYNOPSIS

```
void cd2400IntRx
(
    CD2400_CHAN * pChan
)
```

DESCRIPTION This routine handles the interrupts for all channels for a Receive Data Interrupt.

SEE ALSO *cd2400Sio*

cd2400IntTx()

NAME *cd2400IntTx()* – handle transmitter interrupts

SYNOPSIS

```
void cd2400IntTx
(
    CD2400_CHAN * pChan
)
```

DESCRIPTION This routine handles transmitter interrupts from the MPCC.

SEE ALSO *cd2400Sio*

cdromFsDevCreate()

| | |
|--------------------|---|
| NAME | <i>cdromFsDevCreate()</i> – create a cdromFsLib device |
| SYNOPSIS | <pre>CDROM_VOL_DESC_ID cdromFsDevCreate (char * devName, /* device name */ BLK_DEV * pBlkDev /* ptr to block device */)</pre> |
| DESCRIPTION | This routine creates an instance of a cdromFsLib device in the I/O system. As input, this function requires a pointer to a BLK_DEV structure for the CD-ROM drive on which you want to create a cdromFsLib device. Thus, you should already have called <i>scsiBlkDevCreate()</i> prior to calling <i>cdromFsDevCreate()</i> . |
| RETURNS | CDROM_VOL_DESC_ID, or NULL if error. |
| SEE ALSO | <i>cdromFsLib</i> , <i>cdromFsInit()</i> |

cdromFsInit()

| | |
|--------------------|---|
| NAME | <i>cdromFsInit()</i> – initialize cdromFsLib |
| SYNOPSIS | <pre>STATUS cdromFsInit (void)</pre> |
| DESCRIPTION | This routine initializes cdromFsLib . It must be called exactly once before calling any other routine in cdromFsLib . |
| ERRNO | S_cdromFsLib_ALREADY_INIT |
| RETURNS | OK or ERROR, if cdromFsLib has already been initialized. |
| SEE ALSO | <i>cdromFsLib</i> , <i>cdromFsDevCreate()</i> , <i>iosLib.h</i> |

cdromFsVolConfigShow()

| | |
|--------------------|---|
| NAME | <i>cdromFsVolConfigShow()</i> – show the volume configuration information |
| SYNOPSIS | <pre>VOID cdromFsVolConfigShow (void * arg /* device name or CDROM_VOL_DESC * */)</pre> |
| DESCRIPTION | This routine retrieves the volume configuration for the named cdromFsLib device and prints it to standard output. The information displayed is retrieved from the BLK_DEV structure for the specified device. |
| RETURNS | N/A |
| SEE ALSO | cdromFsLib |

ceil()

| | |
|----------------------|--|
| NAME | <i>ceil()</i> – compute the smallest integer greater than or equal to a specified value (ANSI) |
| SYNOPSIS | <pre>double ceil (double v /* value to find the ceiling of */)</pre> |
| DESCRIPTION | This routine returns the smallest integer greater than or equal to <i>v</i> , in double precision. |
| INCLUDE FILES | math.h |
| RETURNS | The smallest integral value greater than or equal to <i>v</i> , in double precision. |
| SEE ALSO | ansiMath, mathALib |

ceilf()

| | |
|----------------------|--|
| NAME | <i>ceilf()</i> – compute the smallest integer greater than or equal to a specified value (ANSI) |
| SYNOPSIS | <pre>float ceilf (float v /* value to find the ceiling of */)</pre> |
| DESCRIPTION | This routine returns the smallest integer greater than or equal to <i>v</i> , in single precision. |
| INCLUDE FILES | math.h |
| RETURNS | The smallest integral value greater than or equal to <i>v</i> , in single precision. |
| SEE ALSO | mathALib |

cfree()

| | |
|--------------------|--|
| NAME | <i>cfree()</i> – free a block of memory |
| SYNOPSIS | <pre>STATUS cfree (char * pBlock /* pointer to block of memory to free */)</pre> |
| DESCRIPTION | This routine returns to the free memory pool a block of memory previously allocated with <i>calloc()</i> . It is an error to free a memory block that was not previously allocated. |
| RETURNS | OK, or ERROR if the the block is invalid. |
| SEE ALSO | memLib |

chdir()

chdir()

- NAME** *chdir()* – set the current default path
- SYNOPSIS**
- ```
STATUS chdir
(
 char * pathname /* name of the new default path */
)
```
- DESCRIPTION** This routine sets the default I/O path. All relative pathnames specified to the I/O system will be prepended with this pathname. This pathname must be an absolute pathname, i.e., *name* must begin with an existing device name.
- RETURNS** OK, or ERROR if the first component of the pathname is not an existing device.
- SEE ALSO** *ioLib*, *ioDefPathSet()*, *ioDefPathGet()*, *getcwd()*

---

***checkStack()***

- NAME** *checkStack()* – print a summary of each task's stack usage
- SYNOPSIS**
- ```
void checkStack
(
    int taskNameOrId /* task name or task ID; 0 = summarize all */
)
```
- DESCRIPTION** This command displays a summary of stack usage for a specified task, or for all tasks if no argument is given. The summary includes the total stack size (SIZE), the current number of stack bytes used (CUR), the maximum number of stack bytes used (HIGH), and the number of bytes never used at the top of the stack (MARGIN = SIZE - HIGH). For example:

```
-> checkStack tShell
      NAME      ENTRY      TID      SIZE      CUR      HIGH      MARGIN
-----
tShell  _shell      23e1c78  9208      832      3632      5576
```

The maximum stack usage is determined by scanning down from the top of the stack for the first byte whose value is not 0xee. In VxWorks, when a task is spawned, all bytes of a task's stack are initialized to 0xee.

| | |
|---------------------|---|
| DEFICIENCIES | It is possible for a task to write beyond the end of its stack, but not write into the last part of its stack. This will not be detected by <i>checkStack()</i> . |
| RETURNS | N/A |
| SEE ALSO | <i>usrLib</i> , <i>taskSpawn()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , <i>windsh</i> , <i>Tornado User's Guide: Shell</i> |

cisConfigregGet()

| | |
|--------------------|---|
| NAME | <i>cisConfigregGet()</i> – get the PCMCIA configuration register |
| SYNOPSIS | <pre>STATUS cisConfigregGet (int sock, /* socket no. */ int reg, /* configuration register no. */ int * pValue /* content of the register */)</pre> |
| DESCRIPTION | This routine gets that PCMCIA configuration register. |
| RETURNS | OK, or ERROR if it cannot set a value on the PCMCIA chip. |
| SEE ALSO | <i>cisLib</i> |

cisConfigregSet()

| | |
|--------------------|--|
| NAME | <i>cisConfigregSet()</i> – set the PCMCIA configuration register |
| SYNOPSIS | <pre>STATUS cisConfigregSet (int sock, /* socket no. */ int reg, /* register no. */ int value /* content of the register */)</pre> |
| DESCRIPTION | This routine sets the PCMCIA configuration register. |
| RETURNS | OK, or ERROR if it cannot set a value on the PCMCIA chip. |

SEE ALSO **cisLib**

cisFree()

NAME *cisFree()* – free tuples from the linked list

SYNOPSIS `void cisFree`
 (
 `int sock /* socket no. */`
)

DESCRIPTION This routine free tuples from the linked list.

RETURNS N/A

SEE ALSO **cisLib**

cisGet()

NAME *cisGet()* – get information from a PC card's CIS

SYNOPSIS `STATUS cisGet`
 (
 `int sock /* socket no. */`
)

DESCRIPTION This routine gets information from a PC card's CIS, configures the PC card, and allocates resources for the PC card.

RETURNS OK, or ERROR if it cannot get the CIS information, configure the PC card, or allocate resources.

SEE ALSO **cisLib**

cisShow()

NAME *cisShow()* – show CIS information

SYNOPSIS

```
void cisShow
(
    int sock /* socket no. */
)
```

DESCRIPTION This routine shows CIS information.

RETURNS N/A

SEE ALSO *cisShow*

cleanUpStoreBuffer()

NAME *cleanUpStoreBuffer()* – clean up store buffer after a data store error interrupt

SYNOPSIS

```
void cleanUpStoreBuffer
(
    UINT mcntl, /* Value of MMU Control Register */
    BOOL exception /* TRUE if exception, FALSE if int */
)
```

DESCRIPTION This routine cleans up the store buffer after a data store error interrupt. The first queued store is retried. It is logged as either a recoverable or unrecoverable error. Then the store buffer is re-enabled and other queued stores are processed by the store buffer.

RETURNS N/A

SEE ALSO *cacheTiTms390Lib*

clearerr()

clearerr()

| | |
|----------------------|--|
| NAME | <i>clearerr()</i> – clear end-of-file and error flags for a stream (ANSI) |
| SYNOPSIS | <pre>void clearerr (FILE * fp /* stream to clear EOF and ERROR flags for */)</pre> |
| DESCRIPTION | This routine clears the end-of-file and error flags for a specified stream. |
| INCLUDE FILES | stdio.h |
| RETURNS | N/A |
| SEE ALSO | ansiStdio , <i>feof()</i> , <i>ferror()</i> |

clock()

| | |
|----------------------|---|
| NAME | <i>clock()</i> – determine the processor time in use (ANSI) |
| SYNOPSIS | <pre>clock_t clock (void)</pre> |
| DESCRIPTION | This routine returns the implementation's best approximation of the processor time used by the program since the beginning of an implementation-defined era related only to the program invocation. To determine the time in seconds, the value returned by <i>clock()</i> should be divided by the value of the macro CLOCKS_PER_SEC . If the processor time used is not available or its value cannot be represented, <i>clock()</i> returns -1. |
| INCLUDE FILES | time.h |
| RETURNS | ERROR (-1). |
| SEE ALSO | ansiTime |

clock_getres()

| | |
|--------------------|--|
| NAME | <i>clock_getres()</i> – get the clock resolution (POSIX) |
| SYNOPSIS | <pre>int clock_getres (clockid_t clock_id, /* clock ID (always CLOCK_REALTIME) */ struct timespec * res /* where to store resolution */)</pre> |
| DESCRIPTION | This routine gets the clock resolution, in nanoseconds, based on the rate returned by <i>sysClkRateGet()</i> . If <i>res</i> is non-NULL, the resolution is stored in the location pointed to. |
| RETURNS | 0 (OK), or -1 (ERROR) if <i>clock_id</i> is invalid. |
| ERRNO | EINVAL |
| SEE ALSO | <i>clockLib</i> , <i>clock_settime()</i> , <i>sysClkRateGet()</i> , <i>clock_setres()</i> |

clock_gettime()

| | |
|--------------------|---|
| NAME | <i>clock_gettime()</i> – get the current time of the clock (POSIX) |
| SYNOPSIS | <pre>int clock_gettime (clockid_t clock_id, /* clock ID (always CLOCK_REALTIME) */ struct timespec * tp /* where to store current time */)</pre> |
| DESCRIPTION | This routine gets the current value <i>tp</i> for the clock. |
| RETURNS | 0 (OK), or -1 (ERROR) if <i>clock_id</i> is invalid or <i>tp</i> is NULL. |
| ERRNO | EINVAL, EFAULT |
| SEE ALSO | <i>clockLib</i> |

clock_setres()

clock_setres()**NAME** *clock_setres()* – set the clock resolution**SYNOPSIS**

```
int clock_setres
(
    clockid_t      clock_id, /* clock ID (always CLOCK_REALTIME) */
    struct timespec * res    /* resolution to be set */
)
```

DESCRIPTION
This routine sets the clock resolution in the POSIX timers data structures. It does not affect the system clock or auxiliary clocks. This routine should be called to inform the POSIX timers of the new clock resolution if *sysClkRateSet()* has been called after this library has been initialized.

If *res* is non-NULL, the resolution to be set is stored in the location pointed to; otherwise, this routine has no effect.

NOTE Non-POSIX.**RETURNS** 0 (OK), or -1 (ERROR) if *clock_id* is invalid or the resolution is greater than 1 second.**ERRNO** EINVAL**SEE ALSO** *clockLib*, *clock_getres()*, *sysClkRateSet()*

clock_settime()**NAME** *clock_settime()* – set the clock to a specified time (POSIX)**SYNOPSIS**

```
int clock_settime
(
    clockid_t      clock_id, /* clock ID (always CLOCK_REALTIME) */
    const struct timespec * tp /* time to set */
)
```

DESCRIPTION
This routine sets the clock to the value *tp*, which should be a multiple of the clock resolution. If *tp* is not a multiple of the resolution, it is truncated to the next smallest multiple of the resolution.

RETURNS 0 (OK), or -1 (ERROR) if *clock_id* is invalid, *tp* is outside the supported range, or the *tp* nanosecond value is less than 0 or equal to or greater than 1,000,000,000.

ERRNO EINVAL

SEE ALSO *clockLib*, *clock_getres()*

close()

NAME *close()* – close a file

SYNOPSIS

```
STATUS close
(
    int fd /* file descriptor to close */
)
```

DESCRIPTION This routine closes the specified file and frees the file descriptor. It calls the device driver to do the work.

RETURNS The status of the driver close routine, or ERROR if the file descriptor is invalid.

SEE ALSO *ioLib*

closedir()

NAME *closedir()* – close a directory (POSIX)

SYNOPSIS

```
STATUS closedir
(
    DIR * pDir /* pointer to directory descriptor */
)
```

DESCRIPTION This routine closes a directory which was previously opened using *opendir()*. The *pDir* parameter is the directory descriptor pointer that was returned by *opendir()*.

RETURNS OK or ERROR.

SEE ALSO *dirLib*, *opendir()*, *readdir()*, *rewinddir()*

connect()

connect()**NAME** `connect()` – initiate a connection to a socket**SYNOPSIS**

```
STATUS connect
(
    int          s,          /* socket descriptor */
    struct sockaddr * name,  /* addr of socket to connect */
    int          namelen /* length of name, in bytes */
)
```

DESCRIPTION If *s* is a socket of type **SOCK_STREAM**, this routine establishes a virtual circuit between *s* and another socket specified by *name*. If *s* is of type **SOCK_DGRAM**, it permanently specifies the peer to which messages are sent. If *s* is of type **SOCK_RAW**, it specifies the raw socket upon which data is to be sent and received. The *name* parameter specifies the address of the other socket.**RETURNS** OK, or ERROR if the call fails.**SEE ALSO** `sockLib`

connectWithTimeout()**NAME** `connectWithTimeout()` – try to connect over a socket for a specified duration**SYNOPSIS**

```
STATUS connectWithTimeout
(
    int          sock,      /* socket descriptor */
    struct sockaddr * adrs, /* addr of the socket to connect */
    int          adrsLen,  /* length of the socket, in bytes */
    struct timeval * timeVal /* time-out value */
)
```

DESCRIPTION This routine basically the same as `connect()`, except that it lets users specify how long to keep trying to make the new connection.If the *timeVal* is a NULL pointer, this routine acts exactly like `connect()`. If *timeVal* is not NULL, it tries to establish a new connection for the duration of the time specified in *timeVal*. After that time, this routine reports a time-out error if the connection is not established.

RETURNS OK, or ERROR if a connection cannot be established.

SEE ALSO `sockLib`, `connect()`

copy()

NAME `copy()` – copy *in* (or stdin) to *out* (or stdout)

SYNOPSIS

```
STATUS copy
(
    char * in, /* name of file to read (if NULL assume stdin) */
    char * out /* name of file to write (if NULL assume stdout) */
)
```

DESCRIPTION This command copies from the input file to the output file, until an end-of-file is reached.

EXAMPLES The following example displays the file **dog**, found on the default file device:

```
-> copy <dog
```

This example copies from the console to the file **dog**, on device `/ct0/`, until an EOF (default CTRL-D) is typed:

```
-> copy >/ct0/dog
```

This example copies the file **dog**, found on the default file device, to device `/ct0/`:

```
-> copy <dog >/ct0/dog
```

This example makes a conventional copy from the file named **file1** to the file named **file2**:

```
-> copy "file1", "file2"
```

Remember that standard input and output are global; therefore, spawning the first three constructs will not work as expected.

RETURNS OK, or ERROR if *in* or *out* cannot be opened/created, or if there is an error copying from *in* to *out*.

SEE ALSO `usrLib`, `copyStreams()`, `tyEOFSet()`, *VxWorks Programmer's Guide: Target Shell*

copyStreams()

| | |
|--------------------|---|
| NAME | <i>copyStreams()</i> – copy from/to specified streams |
| SYNOPSIS | <pre>STATUS copyStreams (int inFd, /* file descriptor of stream to copy from */ int outFd /* file descriptor of stream to copy to */)</pre> |
| DESCRIPTION | This command copies from the stream identified by <i>inFd</i> to the stream identified by <i>outFd</i> until an end of file is reached in <i>inFd</i> . This command is used by <i>copy()</i> . |
| RETURNS | OK, or ERROR if there is an error reading from <i>inFd</i> or writing to <i>outFd</i> . |
| SEE ALSO | <i>usrLib</i> , <i>copy()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> |

cos()

| | |
|----------------------|---|
| NAME | <i>cos()</i> – compute a cosine (ANSI) |
| SYNOPSIS | <pre>double cos (double x /* angle in radians */)</pre> |
| DESCRIPTION | This routine computes the cosine of <i>x</i> in double precision. The angle <i>x</i> is expressed in radians. |
| INCLUDE FILES | <i>math.h</i> |
| RETURNS | The double-precision cosine of <i>x</i> . |
| SEE ALSO | <i>ansiMath</i> , <i>mathALib</i> |

cosf()

| | |
|----------------------|--|
| NAME | <i>cosf()</i> – compute a cosine (ANSI) |
| SYNOPSIS | <pre>float cosf (float x /* angle in radians */)</pre> |
| DESCRIPTION | This routine returns the cosine of x in single precision. The angle x is expressed in radians. |
| INCLUDE FILES | <code>math.h</code> |
| RETURNS | The single-precision cosine of x . |
| SEE ALSO | <code>mathALib</code> |

cosh()

| | |
|----------------------|--|
| NAME | <i>cosh()</i> – compute a hyperbolic cosine (ANSI) |
| SYNOPSIS | <pre>double cosh (double x /* value to compute the hyperbolic cosine of */)</pre> |
| DESCRIPTION | This routine returns the hyperbolic cosine of x in double precision (IEEE double, 53 bits). A range error occurs if x is too large. |
| INCLUDE FILES | <code>math.h</code> |
| RETURNS | The double-precision hyperbolic cosine of x . Special cases: If x is <code>+INF</code> , <code>-INF</code> , or <code>NaN</code> , <i>cosh()</i> returns x . |
| SEE ALSO | <code>ansiMath</code> , <code>mathALib</code> |

coshf()

coshf()

NAME *coshf()* – compute a hyperbolic cosine (ANSI)

SYNOPSIS

```
float coshf  
(  
    float x /* value to compute the hyperbolic cosine of */  
)
```

DESCRIPTION This routine returns the hyperbolic cosine of *x* in single precision.

INCLUDE FILES **math.h**

RETURNS The single-precision hyperbolic cosine of *x* if the parameter is greater than 1.0, or NaN if the parameter is less than 1.0.

Special cases:
If *x* is +INF, -INF, or NaN, *coshf()* returns *x*.

SEE ALSO **mathALib**

cplusCallNewHandler()

NAME *cplusCallNewHandler()* – call the allocation failure handler (C++)

SYNOPSIS

```
extern void cplusCallNewHandler ()
```

DESCRIPTION This function provides a procedural-interface to the new-handler. It can be used by user-defined new operators to call the current new-handler. This function is specific to VxWorks and may not be available in other C++ environments.

RETURNS N/A

SEE ALSO **cplusLib**

cplusCtors()

| | |
|--------------------|---|
| NAME | <i>cplusCtors()</i> – call static constructors (C++) |
| SYNOPSIS | <pre>extern "C" void cplusCtors (const char * moduleName /* name of loaded module */)</pre> |
| DESCRIPTION | This function is used to call static constructors under the manual strategy (see <i>cplusXtorSet()</i>). <i>moduleName</i> is the name of an object module that was "munched" before loading. If <i>moduleName</i> is 0, then all static constructors, in all modules loaded by the VxWorks module loader, are called. |
| EXAMPLES | <p>The following example shows how to initialize the static objects in modules called "applx.out" and "apply.out".</p> <pre>-> cplusCtors "applx.out" value = 0 = 0x0 -> cplusCtors "apply.out" value = 0 = 0x0</pre> <p>The following example shows how to initialize all the static objects that are currently loaded, with a single invocation of <i>cplusCtors()</i>:</p> <pre>-> cplusCtors value = 0 = 0x0</pre> |
| RETURNS | N/A |
| SEE ALSO | <i>cplusLib</i> , <i>cplusXtorSet()</i> |

cplusCtorsLink()

| | |
|--------------------|--|
| NAME | <i>cplusCtorsLink()</i> – call all linked static constructors (C++) |
| SYNOPSIS | <pre>extern "C" void cplusCtorsLink ()</pre> |
| DESCRIPTION | This function calls constructors for all of the static objects linked with a VxWorks bootable image. When creating bootable applications, this function should be called from <i>usrRoot()</i> to initialize all static objects. Correct operation depends on correctly munching the C++ modules that are linked with VxWorks. |

RETURNS N/A

SEE ALSO `cplusLib`

cplusDemanglerSet()

NAME `cplusDemanglerSet()` – change C++ demangling mode (C++)

SYNOPSIS

```
extern "C" void cplusDemanglerSet
(
    int mode
)
```

DESCRIPTION This command sets the C++ demangling mode to *mode*. The default mode is 2. There are three demangling modes, *complete*, *terse*, and *off*. These modes are represented by numeric codes:

| Mode | Code |
|-------------|-------------|
| off | 0 |
| terse | 1 |
| complete | 2 |

In complete mode, when C++ function names are printed, the class name (if any) is prefixed and the function's parameter type list is appended.

In terse mode, only the function name is printed. The class name and parameter type list are omitted.

In off mode, the function name is not demangled.

EXAMPLES The following example shows how one function name would be printed under each demangling mode:

| Mode | Printed symbol |
|-------------|--|
| off | <code>_member__5classFPFI_PvPFPv_v</code> |
| terse | <code>_member</code> |
| complete | <code>foo::_member(void* (*)(long),void (*)(void*))</code> |

RETURNS N/A

SEE ALSO `cplusLib`

cplusDtors()

| | |
|--------------------|---|
| NAME | <i>cplusDtors()</i> – call static destructors (C++) |
| SYNOPSIS | <pre>extern "C" void cplusDtors (const char * moduleName)</pre> |
| DESCRIPTION | This function is used to call static destructors under the manual strategy (see <i>cplusXtorSet()</i>). <i>moduleName</i> is the name of an object module that was “munched” before loading. If <i>moduleName</i> is 0, then all static destructors, in all modules loaded by the VxWorks module loader, are called. |
| EXAMPLES | <p>The following example shows how to destroy the static objects in modules called "applx.out" and "apply.out":</p> <pre>-> cplusDtors "applx.out" value = 0 = 0x0 -> cplusDtors "apply.out" value = 0 = 0x0</pre> <p>The following example shows how to destroy all the static objects that are currently loaded, with a single invocation of <i>cplusDtors()</i>:</p> <pre>-> cplusDtors value = 0 = 0x0</pre> |
| RETURNS | N/A |
| SEE ALSO | <i>cplusLib</i> , <i>cplusXtorSet()</i> |

cplusDtorsLink()

| | |
|--------------------|--|
| NAME | <i>cplusDtorsLink()</i> – call all linked static destructors (C++) |
| SYNOPSIS | <pre>extern "C" void cplusDtorsLink ()</pre> |
| DESCRIPTION | This function calls destructors for all of the static objects linked with a VxWorks bootable image. When creating bootable applications, this function should be called during system shutdown to decommission all static objects. Correct operation depends on correctly munching the C++ modules that are linked with VxWorks. |

RETURNS N/A

SEE ALSO `cplusLib`

cplusLibInit()

NAME `cplusLibInit()` – initialize the C++ library (C++)

SYNOPSIS `extern "C" STATUS cplusLibInit (void)`

DESCRIPTION This routine initializes the C++ library and forces all C++ run-time support to be linked with the bootable VxWorks image. If the configuration macro `INCLUDE_CPLUS` is defined, `cplusLibInit()` is called automatically from the root task, `usrRoot()`, in `usrConfig.c`.

RETURNS OK or ERROR.

SEE ALSO `cplusLib`

cplusXtorSet()

NAME `cplusXtorSet()` – change C++ static constructor calling strategy (C++)

SYNOPSIS `extern "C" void cplusXtorSet
(
 int strategy
)`

DESCRIPTION This command sets the C++ static constructor calling strategy to *strategy*. The default strategy is 0.

There are two static constructor calling strategies: *automatic* and *manual*. These modes are represented by numeric codes:

| Strategy | Code |
|-----------------|-------------|
| manual | 0 |
| automatic | 1 |

Under the manual strategy, a module's static constructors and destructors are called by *cplusCtors()* and *cplusDtors()*, which are themselves invoked manually.

Under the automatic strategy, a module's static constructors are called as a side-effect of loading the module using the VxWorks module loader. A module's static destructors are called as a side-effect of unloading the module.

| | |
|-----------------|---|
| NOTE | The manual strategy is applicable only to modules that are loaded by the VxWorks module loader. Static constructors and destructors contained by modules linked with the VxWorks image are called using <i>cplusCtorsLink()</i> and <i>cplusDtorsLink()</i> . |
| RETURNS | N/A |
| SEE ALSO | <i>cplusLib</i> |

cpmattach()

NAME *cpmattach()* – publish the **cpm** network interface and initialize the driver

SYNOPSIS

```

STATUS cpmattach
(
    int          unit,      /* unit number */
    SCC *        pScc,      /* address of SCC parameter RAM */
    SCC_REG *    pSccReg,   /* address of SCC registers */
    VOIDFUNCPTR * ivec,    /* interrupt vector offset */
    SCC_BUF *    txBdBase,  /* transmit buffer descriptor base address */
    SCC_BUF *    rxBdBase,  /* receive buffer descriptor base address */
    int          txBdNum,   /* number of transmit buffer descriptors */
    int          rxBdNum,   /* number of receive buffer descriptors */
    UINT8 *      bufBase   /* address of memory pool; NONE = malloc it */
)

```

DESCRIPTION The routine publishes the **cpm** interface by filling in a network Interface Data Record (IDR) and adding this record to the system's interface list.

The SCC shares a region of memory with the driver. The caller of this routine can specify the address of a shared, non-cacheable memory region with *bufBase*. If this parameter is *NONE*, the driver obtains this memory region by calling *cacheDmaMalloc()*.

Non-cacheable memory space is important for cases where the SCC is operating with a processor that has a data cache.

Once non-cacheable memory is obtained, this routine divides up the memory between the various buffer descriptors (BDs). The number of BDs can be specified by *txBdNum* and *rxBdNum*, or if *NULL*, a default value of 32 BDs will be used. Additional buffers are

reserved as receive loaner buffers. The number of loaner buffers is the lesser of *rxBdNum* and a default value of 16.

The user must specify the location of the transmit and receive BDs in the CPU's dual-ported RAM. *txBdBase* and *rxBdBase* give the base address of the BD rings. Each BD uses 8 bytes. Care must be taken so that the specified locations for Ethernet BDs do not conflict with other dual-ported RAM structures.

Up to four individual device units are supported by this driver. Device units may reside on different processor chips, or may be on different SCCs within a single CPU.

Before this routine returns, it calls *cpmReset()* and *cpmInit()* to configure the Ethernet controller, and connects the interrupt vector *ivec*.

RETURNS OK or ERROR.

SEE ALSO *if_cpm*, *ifLib*, *Motorola MC68360 User's Manual*, *Motorola MPC821 and MPC860 User's Manual*

cpmStartOutput()

NAME *cpmStartOutput()* – output packet to network interface device

SYNOPSIS

```
#ifdef BSD43_DRIVER LOCAL void cpmStartOutput
(
    int unit /* unit number */
)
```

DESCRIPTION *cpmStartOutput()* takes a packet from the network interface output queue, copies the mbuf chain into an interface buffer, and sends the packet over the interface. *etherOutputHookRtns* are supported.

Collision stats are collected in this routine from previously sent BDs. These BDs will not be examined until after the transmitter has cycled the ring, coming upon the BD after it has been sent. Thus, collision stat collection will be delayed a full cycle through the Tx ring.

This routine is called from several possible threads. Each one will be described below.

The first, and most common thread, is when a user task requests the transmission of data. Under BSD 4.3, this will cause *cpmOutput()* to be called, which calls *ether_output()*, which usually calls this routine. This routine will not be called if *ether_output()* finds that our interface output queue is full. In this very rare case, the outgoing data will be thrown out. BSD 4.4 uses a slightly different model in which the generic *ether_output()* routine is called directly, followed by a call to this routine.

The second thread is when a transmitter error occurs that causes a TXE event interrupt. This happens for the following errors: transmitter underrun, retry limit reached, late collision, and heartbeat error. The ISR sets the txStop flag to stop the transmitter until the errors are serviced. These events require a RESTART command of the transmitter, which occurs in the *cpmTxRestart()* routine. After the transmitter is restarted, *cpmTxRestart()* does a netJobAdd of *cpmStartOutput()* to send any packets left in the interface output queue. Thus, the second thread executes in the context of *netTask()*.

The third, and most unlikely, thread occurs when this routine is executing and it runs out of free Tx BDs. In this case, this routine turns on transmit interrupt and exits. When the next BD is actually sent, an interrupt occurs. The ISR does a netJobAdd of *cpmStartOutput()* to continue sending packets left in the interface output queue. Once again, we find ourselves executing in the context of *netTask()*.

RETURNS N/A

SEE ALSO *if_cpm*

cpsr()

NAME *cpsr()* – return the contents of the current processor status register (ARM)

SYNOPSIS

```
int cpsr
(
    int taskId /* task ID, 0 means default task */
)
```

DESCRIPTION This command extracts the contents of the status register from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.

RETURNS The contents of the current processor status register.

SEE ALSO *dbgArchLib*, *VxWorks Programmer's Guide: Debugging*

creat()

creat()**NAME** *creat()* – create a file**SYNOPSIS**

```
int creat
(
    const char * name, /* name of the file to create */
    int         flag /* O_RDONLY, O_WRONLY, or O_RDWR */
)
```

DESCRIPTION This routine creates a file called *name* and opens it with a specified *flag*. This routine determines on which device to create the file; it then calls the create routine of the device driver to do most of the work. Therefore, much of what transpires is device/driver-dependent.

The parameter *flag* is set to **O_RDONLY** (0), **O_WRONLY** (1), or **O_RDWR** (2) for the duration of time the file is open. To create NFS files with a UNIX chmod-type file mode, call *open()* with the file mode specified in the third argument.

NOTE For more information about situations when there are no file descriptors available, see the manual entry for *iosInit()*.**RETURNS** A file descriptor number, or ERROR if a filename is not specified, the device does not exist, no file descriptors are available, or the driver returns ERROR.**SEE ALSO** *ioLib*, *open()*

cret()**NAME** *cret()* – continue until the current subroutine returns**SYNOPSIS**

```
STATUS cret
(
    int task /* task to continue, 0 = default */
)
```

DESCRIPTION This routine places a breakpoint at the return address of the current subroutine of a specified task, then continues execution of that task.

To execute, enter:

```
-> cret [task]
```

If *task* is omitted or zero, the last task referenced is assumed.

When the breakpoint is hit, information about the task will be printed in the same format as in single-stepping. The breakpoint is automatically removed when hit, or if the task hits another breakpoint first.

RETURNS OK, or ERROR if there is no such task or the breakpoint table is full.

SEE ALSO **dbgLib**, **so()**, *VxWorks Programmer's Guide: Shell*, **windsh**, *Tornado User's Guide: Shell*

csAttach()

NAME **csAttach()** – publish the **cs** network interface and initialize the driver.

SYNOPSIS

```
STATUS csAttach
(
    int    unit,          /* unit number */
    int    ioAddr,       /* base IO address */
    int    intVector,    /* interrupt vector, or zero */
    int    intLevel,     /* interrupt level */
    int    memAddr,      /* base memory address */
    int    mediaType,    /* 0: Autodetect 1: AUI 2: BNC 3: RJ45 */
    int    configFlags,  /* configuration flag */
    char * pEnetAddr     /* ethernet address */
)
```

DESCRIPTION This routine is a major entry point to this network interface driver and is called only once per operating system reboot by the operating system startup code. This routine is called before the **csInit()** routine.

This routine takes passed-in configuration parameters and parameters from the EEPROM and fills in the instance global variables in the **cs_softc** structure; these variables are later used by **csChipInit()**. **csAttach()** connects the interrupt handler **csIntr()** to the specified interrupt vector, initializes the 8259 PIC, and resets the CS8900 chip.

Finally, **csAttach()** calls the **ether_attach()** routine to fill in the **ifnet** structure and attach this network interface driver to the system. The driver's main entry points (**csInit()**, **csIoctl()**, **csOutput()**, **csReset()**) are made visible to the protocol stack.

See the reference page for **if_cs** for a detailed description of the configuration flags.

RETURNS OK or ERROR.

SEE ALSO **if_cs**

csShow()

csShow()

| | |
|--------------------|---|
| NAME | <i>csShow()</i> – shows statistics for the cs network interface |
| SYNOPSIS | <pre>void csShow (int unit, /* interface unit */ BOOL zap /* zero totals */)</pre> |
| DESCRIPTION | <p>This routine displays statistics about the cs Ethernet network interface. It has two parameters:</p> <p><i>unit</i> interface unit; should be 0.</p> <p><i>zap</i> if 1, all collected statistics are cleared to zero.</p> |
| RETURNS | N/A |
| SEE ALSO | <i>if_cs</i> |

ctime()

| | |
|----------------------|--|
| NAME | <i>ctime()</i> – convert time in seconds into a string (ANSI) |
| SYNOPSIS | <pre>char * ctime (const time_t * timer /* calendar time in seconds */)</pre> |
| DESCRIPTION | <p>This routine converts the calendar time pointed to by <i>timer</i> into local time in the form of a string. It is equivalent to:</p> <pre>asctime (localtime (timer));</pre> <p>This routine is not reentrant. For a reentrant version, see <i>ctime_r()</i>.</p> |
| INCLUDE FILES | time.h |
| RETURNS | The pointer returned by <i>asctime()</i> with local broken-down time as the argument. |
| SEE ALSO | ansiTime, asctime(), localtime() |

ctime_r()

| | |
|----------------------|--|
| NAME | <i>ctime_r()</i> – convert time in seconds into a string (POSIX) |
| SYNOPSIS | <pre>char * ctime_r (const time_t * timer, /* calendar time in seconds */ char * asctimeBuf, /* buffer to contain the string */ size_t * buflen /* size of the buffer */)</pre> |
| DESCRIPTION | <p>This routine converts the calendar time pointed to by <i>timer</i> into local time in the form of a string. It is equivalent to:</p> <pre> asctime (localtime (timer));</pre> <p>This routine is the POSIX re-entrant version of <i>ctime()</i>.</p> |
| INCLUDE FILES | <code>time.h</code> |
| RETURNS | The pointer returned by <i>asctime()</i> with local broken-down time as the argument. |
| SEE ALSO | <i>ansiTime</i> , <i>asctime()</i> , <i>localtime()</i> |

d()

| | |
|--------------------|---|
| NAME | <i>d()</i> – display memory |
| SYNOPSIS | <pre>void d (void * adrs, /* address to display (if 0, display next block */ int nunits, /* number of units to print (if 0, use default) */ int width /* width of displaying unit (1, 2, 4, 8) */)</pre> |
| DESCRIPTION | <p>This command displays the contents of memory, starting at <i>adrs</i>. If <i>adrs</i> is omitted or zero, <i>d()</i> displays the next memory block, starting from where the last <i>d()</i> command completed.</p> <p>Memory is displayed in units specified by <i>width</i>. If <i>nunits</i> is omitted or zero, the number of units displayed defaults to last use. If <i>nunits</i> is non-zero, that number of units is displayed and that number then becomes the default. If <i>width</i> is omitted or zero, it</p> |

d0()

defaults to the previous value. If *width* is an invalid number, it is set to 1. The valid values for *width* are 1, 2, 4, and 8. The number of units *d()* displays is rounded up to the nearest number of full lines.

RETURNS N/A

SEE ALSO **usrLib**, **m()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

d0()

NAME *d0()* – return the contents of register **d0** (also **d1** – **d7**) (MC680x0)

SYNOPSIS

```
int d0
(
    int taskId /* task ID, 0 means default task */
)
```

DESCRIPTION This command extracts the contents of register **d0** from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.

Similar routines are provided for all data registers (**d0** – **d7**): *d0()* – *d7()*.

RETURNS The contents of register **d0** (or the requested register).

SEE ALSO **dbgArchLib**, *VxWorks Programmer's Guide: Target Shell*

dbgBpTypeBind()

NAME *dbgBpTypeBind()* – bind a breakpoint handler to a breakpoint type (MIPS R3000, R4000)

SYNOPSIS

```
STATUS dbgBpTypeBind
(
    int    bpType, /* breakpoint type */
    FUNCPTR routine /* function to bind */
)
```

DESCRIPTION Dynamically bind a breakpoint handler to breakpoints of type 0 – 7. By default only breakpoints of type zero are handled with the function *dbgBreakpoint()* (see **dbgLib**).

Other types may be used for Ada stack overflow or other such functions. The installed handler must take the same parameters as *excExcHandle()* (see *excLib*).

RETURNS OK, or ERROR if *bpType* is out of bounds.

SEE ALSO *dbgArchLib*, *dbgLib*, *excLib*

dbgHelp()

NAME *dbgHelp()* – display debugging help menu

SYNOPSIS `void dbgHelp (void)`

DESCRIPTION This routine displays a summary of **dbgLib** utilities with a short description of each, similar to the following:

| | | |
|----------------------|--|---|
| <code>dbgHelp</code> | | Print this list |
| <code>dbgInit</code> | | Install debug facilities |
| <code>b</code> | | Display breakpoints |
| <code>b</code> | <code>addr[,task[,count]]</code> | Set breakpoint |
| <code>e</code> | <code>addr[,eventNo[,task[,func[,arg]]]]</code> | Set eventpoint (WindView) |
| <code>bd</code> | <code>addr[,task]</code> | Delete breakpoint |
| <code>bdall</code> | <code>[task]</code> | Delete all breakpoints |
| <code>c</code> | <code>[task[,addr[,addr1]]]</code> | Continue from breakpoint |
| <code>cret</code> | <code>[task]</code> | Continue to subroutine return |
| <code>s</code> | <code>[task[,addr[,addr1]]]</code> | Single step |
| <code>so</code> | <code>[task]</code> | Single step/step over subroutine |
| <code>l</code> | <code>[adr[,nInst]]</code> | List disassembled memory |
| <code>tt</code> | <code>[task]</code> | Do stack trace on task |
| <code>bh</code> | <code>addr[,access[,task[,count[,quiet]]]</code> | set hardware breakpoint (if supported by the architecture) |

RETURNS N/A

SEE ALSO *dbgLib*, *VxWorks Programmer's Guide: Target Shell*

dbgInit()

dbgInit()

| | |
|--------------------|---|
| NAME | <i>dbgInit()</i> – initialize the local debugging package |
| SYNOPSIS | <code>STATUS dbgInit (void)</code> |
| DESCRIPTION | This routine initializes the local debugging package and enables the basic breakpoint and single-step functions. This routine also enables the shell abort function, CTRL-C. |
| NOTE | The debugging package should be initialized before any debugging routines are used. If the configuration macro <code>INCLUDE_DEBUG</code> is defined, <i>dbgInit()</i> is called by the root task, <i>usrRoot()</i> , in <code>usrConfig.c</code> . |
| RETURNS | OK, always. |
| SEE ALSO | <i>dbgLib</i> , <i>VxWorks Programmer's Guide: Target Shell</i> |

dcattach()

| | |
|--------------------|--|
| NAME | <i>dcattach()</i> – publish the <code>dc</code> network interface. |
| SYNOPSIS | <pre>STATUS dcattach (int unit, /* unit number */ ULONG devAdrs, /* device I/O address */ int ivec, /* interrupt vector */ int ilevel, /* interrupt level */ char * memAdrs, /* address of memory pool (-1 = malloc it) */ ULONG memSize, /* only used if memory pool is NOT malloc()'d */ int memWidth, /* byte-width of data (-1 = any width) */ ULONG pciMemBase, /* main memory base as seen from PCI bus */ int dcOpMode /* mode of operation */) </pre> |
| DESCRIPTION | This routine publishes the <code>dc</code> interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state. The <i>unit</i> parameter is used to specify the device unit to initialize. |

The *devAdrs* is used to specify the I/O address base of the device.

The *ivec* parameter is used to specify the interrupt vector associated with the device interrupt.

The *ilevel* parater is used to specify the level of the interrupt which the device would use.

The *memAdrs* parameter can be used to specify the location of the memory that will be shared between the driver and the device. The value NONE is used to indicate that the driver should obtain the memory.

The *memSize* parameter is valid only if the *memAdrs* parameter is not set to NONE, in which case *memSize* indicates the size of the provided memory region.

The *memWidth* parameter sets the memory pool's data port width (in bytes); if it is NONE, any data width is used.

The *pciMemBase* parameter defines the main memory base as seen from PCI bus.

The *dcOpMode* parameter defines the mode in which the device should be operational.

| | |
|-----------------|---|
| BUGS | To zero out DEC 21x4x data structures, this routine uses <i>bzero()</i> , which ignores the <i>memWidth</i> specification and uses any size data access to write to memory. |
| RETURNS | OK or ERROR. |
| SEE ALSO | <i>if_dc</i> |

dcCsrShow()

| | |
|--------------------|--|
| NAME | <i>dcCsrShow()</i> – display dec 21040/21140 status registers 0 thru 15 |
| SYNOPSIS | <pre>int dcCsrShow (int unit)</pre> |
| DESCRIPTION | Display the 16 registers of the DEC 21140 device on the console. Each register is printed in hexadecimal format. |
| RETURNS | N/A. |
| SEE ALSO | <i>if_dc</i> |

dcReadAllRom()

NAME *dcReadAllRom()* – read entire serial rom

SYNOPSIS

```
void dcReadAllRom
(
    ULONG devAdrs, /* device base I/O address */
    UCHAR * buffer, /* destination bufferr */
    int cnt /* Amount to extract in bytes */
)
```

DESCRIPTION Function to read all of serial rom and store the data in the data structure passed to the function. The count value will indicate how much of the serial rom to read. The routine with also swap the the bytes as the come in.

RETURNS N/A.

SEE ALSO *if_dc*

dcViewRom()

NAME *dcViewRom()* – display lines of serial ROM for dec21140

SYNOPSIS

```
int dcViewRom
(
    ULONG devAdrs, /* device base I/O address */
    UCHAR lineCnt, /* Serial ROM line Number */
    int cnt /* Amount to display */
)
```

RETURNS Number of bytes displayed.

SEE ALSO *if_dc*

dec21x4xEndLoad()

| | |
|--------------------|--|
| NAME | <i>dec21x4xEndLoad()</i> – initialize the driver and device |
| SYNOPSIS | <pre>END_OBJ * dec21x4xEndLoad (char * initStr /* String to be parse by the driver. */)</pre> |
| DESCRIPTION | <p>This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the <i>initString</i>.</p> <p>This routine can be called in two modes. If it is called with an empty, but allocated string then it places the name of this device (i.e. dc) into the <i>initString</i> and returns 0.</p> <p>If the string is allocated then the routine attempts to perform its load functionality.</p> |
| RETURNS | An END object pointer or NULL on error or 0 and the name of the device if the <i>initString</i> was NULL. |
| SEE ALSO | <i>dec21x4xEnd</i> |

dec21x40EndLoad()

| | |
|--------------------|--|
| NAME | <i>dec21x40EndLoad()</i> – initialize the driver and device |
| SYNOPSIS | <pre>END_OBJ* dec21x40EndLoad (char* initStr /* String to be parse by the driver. */)</pre> |
| DESCRIPTION | <p>This routine initializes the driver and the device to an operational state. All of the device-specific parameters are passed in the <i>initStr</i>. If this routine is called with an empty but allocated string, it puts the name of this device (that is, "dc") into the <i>initStr</i> and returns 0. If the string is allocated but not empty, this routine tries to load the device.</p> |
| RETURNS | An END object pointer or NULL on error. |
| SEE ALSO | <i>dec21x40End</i> |

dec21x40PhyLinkPoll()

NAME *dec21x40PhyLinkPoll()* – Poll the PHY for link status

SYNOPSIS

```
UINT dec21x40PhyLinkPoll
(
    DRV_CTRL * pDrvCtrl,
    UINT      linkTry
)
```

RETURNS number of poll iterations remaining when link became active

SEE ALSO *dec21x40End*

dec21140SromWordRead()

NAME *dec21140SromWordRead()* – read two bytes from the serial ROM

SYNOPSIS

```
USHORT dec21140SromWordRead
(
    DRV_CTRL * pDrvCtrl,
    UCHAR     lineCnt /* Serial ROM line Number */
)
```

DESCRIPTION This routine returns the two bytes of information that is associated with it the specified ROM line number. This will later be used by the *dec21140GetEthernetAdr()* function. It can also be used to review the ROM contents itself. The function must first send some initial bit patterns to the CSR9 that contains the Serial ROM Control bits. Then the line index into the ROM is evaluated bit-by-bit to program the ROM. The 2 bytes of data are extracted and processed into a normal pair of bytes.

RETURNS Value from ROM or ERROR.

SEE ALSO *dec21x40End*

devs()

| | |
|--------------------|--|
| NAME | <i>devs()</i> – list all system-known devices |
| SYNOPSIS | <code>void devs (void)</code> |
| DESCRIPTION | This command displays a list of all devices known to the I/O system. |
| RETURNS | N/A |
| SEE ALSO | <code>usrLib</code> , <code>iosDevShow()</code> , <i>VxWorks Programmer's Guide: Target Shell</i> , <code>windsh</code> , <i>Tornado User's Guide: Shell</i> |

dhcpcBind()

| | |
|--------------------|---|
| NAME | <i>dhcpcBind()</i> – obtain a set of network configuration parameters with DHCP |
| SYNOPSIS | <pre>STATUS dhcpcBind (void * pCookie, /* identifier returned by dhcpcInit() */ BOOL syncFlag /* synchronous or asynchronous execution */)</pre> |
| DESCRIPTION | <p>This routine initiates a DHCP negotiation according to the process described in RFC 1541. The <i>pCookie</i> argument contains the return value of an earlier <i>dhcpcInit()</i> call and is used to identify a particular lease.</p> <p>The <i>syncFlag</i> parameter specifies whether the DHCP negotiation started by this routine will execute synchronously or asynchronously. An asynchronous execution will return after starting the DHCP negotiation, but a synchronous execution will only return once the negotiation process completes.</p> <p>When a new lease is established, any event hook provided for the lease will be called to process the configuration parameters. The hook is also called when the lease expires or the negotiation process fails. The results of an asynchronous DHCP negotiation are not available unless an event hook is installed.</p> <p>If automatic configuration of the underlying network interface was specified during the lease initialization, this routine will prevent all higher-level protocols from accessing the underlying network interface used during the initial lease negotiation until that process is complete. In addition, any addressing information obtained will be applied to that</p> |

network interface, which will remain disabled if the initial negotiation fails. Finally, the interface will be disabled if the lease expires.

| | |
|-----------------|--|
| NOTE | If the DHCP client is used to obtain the VxWorks boot parameters, this routine is called automatically during system startup using the automatic reconfiguration. Therefore, any calls to this routine which use the network boot device for message transfer when the DHCP client was used at boot time must not request automatic reconfiguration during initialization. Otherwise, the resulting lease settings will conflict with the configuration maintained by the lease established during system startup. |
| RETURNS | OK if routine completes, or ERROR otherwise. |
| ERRNO | S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED, S_dhcpcLib_BAD_OPTION |
| SEE ALSO | dhcpcLib |

dhcpcBootBind()

| | |
|--------------------|--|
| NAME | <i>dhcpcBootBind()</i> – initialize the network with DHCP at boot time |
| SYNOPSIS | STATUS <i>dhcpcBootBind</i> (void) |
| DESCRIPTION | This routine performs the client side of a DHCP negotiation according to RFC 1541. The negotiation uses the network device specified with the initialization call. The addressing information retrieved is applied to that network device. Because the boot image is replaced by the downloaded target image, the resulting lease cannot be renewed. Therefore, the minimum lease length specified by DHCPC_MIN_LEASE must be set so that the target image has sufficient time to download and begin monitoring the lease. This routine is called automatically by the boot program when INCLUDE_DHCPC is defined and the automatic configuration option is set in the boot flags. |
| RETURNS | OK if negotiation is successful, or ERROR otherwise. |
| ERRNO | N/A |
| SEE ALSO | dhcpcBootLib |

dhcpcBootInit()

| | |
|--------------------|---|
| NAME | <i>dhcpcBootInit()</i> – set up the DHCP client parameters and data structures |
| SYNOPSIS | <pre>STATUS dhcpcBootInit (struct ifnet * pIf /* network device used by client */)</pre> |
| DESCRIPTION | This routine creates any necessary data structures and sets the client's option request list to retrieve a subnet mask and broadcast address for the network interface indicated by <i>pIf</i> . The routine is executed automatically by the boot program when <code>INCLUDE_DHCP</code> is defined and the automatic configuration option is set in the boot flags. The network interface specified by <i>pIf</i> is used to transmit and receive all DHCP messages during the lease negotiation. That interface must be capable of sending broadcast messages. Currently, only Ethernet devices and the shared-memory network drivers are supported. |
| ERRNO | N/A |
| RETURNS | OK, or ERROR if could not initialize. |
| SEE ALSO | <i>dhcpcBootLib</i> |

dhcpcBootOptionSet()

| | |
|--------------------|--|
| NAME | <i>dhcpcBootOptionSet()</i> – add an option to the option request list |
| SYNOPSIS | <pre>STATUS dhcpcBootOptionSet (int option, /* RFC 1533 tag of desired option */ long value, /* numeric value for option */ long length, /* length of data (if any) or 0 if unused */ char * pData /* option data, or NULL if none */)</pre> |
| DESCRIPTION | This routine sets most client-to-server transmission options for a lease established by the boot program. The <i>option</i> parameter specifies an option tag as defined in RFC 1533 and the updates published in the Internet Draft of November 1996. The boot program automatically sets all necessary options for target configuration. This routine is only provided to support special circumstances in which additional options are necessary. |

Any options requested with this routine may be retrieved after the runtime image has started. For a listing of defined aliases for the known option tags, see ***dhcpc/dhcp.h***. This routine cannot set the options associated with the following tags:

_DHCP_PAD_TAG
_DHCP_OPT_OVERLOAD_TAG
_DHCP_MSGTYPE_TAG
_DHCP_SERVER_ID_TAG
_DHCP_REQ_LIST_TAG
_DHCP_MAXMSGSIZE_TAG
_DHCP_END_TAG

Most options only require specification of the appropriate tag in the *option* parameter. In those cases, the ***dhcpcBootOptionSet()*** call adds the specified option tag to the option request list, if possible. However, some options require additional information. The tags for these options are:

_DHCP_VENDOR_SPEC_TAG
_DHCP_REQUEST_IPADDR_TAG
_DHCP_LEASE_TIME_TAG
_DHCP_ERR_MSG_TAG
_DHCP_CLASS_ID_TAG
_DHCP_CLIENT_ID_TAG

The ***_DHCP_LEASE_TIME_TAG*** and ***_DHCP_CLIENT_ID_TAG*** options each require a *value* parameter. For ***_DHCP_LEASE_TIME_TAG***, *value* specifies the desired lease length. For ***_DHCP_CLIENT_ID_TAG***, *value* specifies the type for a type/value pair. No other options use this parameter.

The ***_DHCP_VENDOR_SPEC_TAG***, ***_DHCP_CLASS_ID_TAG***, and ***_DHCP_CLIENT_ID_TAG***, tags each require a value for the *length* parameter to specify the number of bytes of data provided. No other options use this parameter.

Use the *data* parameter with the following option tags:

_DHCP_VENDOR_SPEC_TAG
The *data* parameter points to a list of "length" bytes of options in the format specified by RFC 1533.

_DHCP_REQUEST_IPADDR_TAG
The *data* parameter points to the string representation of the desired Internet address for the client.

_DHCP_ERRMSG_TAG
The *data* parameter points to the error message to send to the server when releasing the current IP address.

_DHCP_CLASS_ID_TAG
The *data* parameter points to *length* bytes used as the value for the vendor class identifier.

_DHCP_CLIENT_ID_TAG

The *data* parameter points to *length* bytes used as the value of a type/value pair.

-

The data parameter should be NULL for all other options.

- NOTE** With the exception of the **_DHCP_ERR_MSG_TAG** option, the DHCP specification forbids changing options after a lease has been established. Therefore, this routine should not be used after the *dhcpcBootBind()* call. Changing any option other than the error message at that point could have unpredictable results.
- RETURNS** OK if option set successfully, or ERROR if option is invalid or storage failed.
- ERRNO** N/A
- SEE ALSO** *dhcpcBootLib*

dhcpcCacheHookAdd()

NAME *dhcpcCacheHookAdd()* – add a routine to store and retrieve lease data

SYNOPSIS

```
STATUS dhcpcCacheHookAdd
(
    FUNCPTR pCacheHookRtn /* routine to store/retrieve lease data */
)
```

DESCRIPTION This routine adds a hook routine that is called at the bound state (to store the lease data) and during the INIT_REBOOT state (to re-use the parameters if the lease is still active). The calling sequence of the input hook routine is:

```
STATUS dhcpcCacheHookRtn
(
    int command, /* requested cache operation */
    unsigned long *pTimeStamp, /* lease timestamp data */
    int *pDataLen, /* length of data to access */
    char *pBuffer /* pointer to data buffer */
)
```

The hook routine should return OK if the requested operation is completed successfully, or ERROR otherwise. All the supplied pointers reference memory locations that are reused upon return from the hook. The hook routine must copy the data elsewhere.

NOTE The setting of the cache hook routine during a *dhcpcInit()* call is recorded and used by the resulting lease throughout its lifetime. Since the hook routine is intended to store a single lease record, a separate hook routine should be specified before the *dhcpcInit()* call for each lease which will re-use its parameters across reboots.

IMPLEMENTATION The *command* parameter specifies one of the following operations:

DHCP_CACHE_WRITE

Save the indicated data. The write operation must preserve the value referenced by *pTimeStamp* and the contents of *pBuffer*. The *pDataLen* parameter indicates the number of bytes in that buffer.

DHCP_CACHE_READ

Restore the saved data. The read operation must copy the data from the most recent write operation into the location indicated by *pBuffer*, set the contents of *pDataLen* to the amount of data provided, and store the corresponding timestamp value in *pTimeStamp*.

-
The read operation has very specific requirements. On entry, the value referenced by *pDataLen* indicates the maximum buffer size available at *pBuffer*. If the amount of data stored by the previous write exceeds this value, the operation must return ERROR. A read must also return ERROR if the saved timestamp value is 0. Finally, the read operation must return ERROR if it is unable to retrieve all the data stored by the write operation or if the previous write was unsuccessful.

DHCP_CACHE_ERASE

Ignore all stored data. Following this operation, subsequent read operations must return ERROR until new data is written. All parameters except *command* are NULL.

RETURNS OK, always.

ERRNO N/A

SEE ALSO ***dhcpcLib***

dhcpcCacheHookDelete()

NAME *dhcpcCacheHookDelete()* – delete a lease data storage routine

SYNOPSIS `STATUS dhcpcCacheHookDelete (void)`

DESCRIPTION This routine deletes the hook used to store lease data, preventing re-use of the configuration parameters across system reboots for all subsequent lease attempts.

Currently active leases will continue to use the routine specified before the lease initialization.

RETURNS OK, always.

ERRNO N/A

SEE ALSO **dhcpcLib**

dhcpcEventHookAdd()

NAME *dhcpcEventHookAdd()* – add a routine to handle configuration parameters

SYNOPSIS

```
STATUS dhcpcEventHookAdd
(
    void * pCookie, /* identifier returned by dhcpcInit() */
    FUNCPTR pEventHook /* routine to handle lease parameters */
)
```

DESCRIPTION This routine installs a hook routine to handle changes in the configuration parameters provided for the lease indicated by *pCookie*. The hook provides an alternate configuration method for DHCP leases and uses the following interface:

```
void dhcpcEventHookRtn
(
    int leaseEvent, /* new or expired parameters */
    void * pCookie /* lease identifier from dhcpcInit() */
)
```

The routine is called with the *leaseEvent* parameter set to **DHCP_LEASE_NEW** whenever a lease is successfully established. The **DHCP_LEASE_NEW** event does not occur when a lease is renewed by the same DHCP server, since the parameters do not change in that case. However, it does occur if the client rebinds to a different DHCP server. The **DHCP_LEASE_INVALID** event indicates that the configuration parameters for the corresponding lease may no longer be used. That event occurs when a lease expires or a renewal or verification attempt fails, and coincides with re-entry into the initial state of the negotiation process.

If the lease initialization specified automatic configuration of the corresponding network interface, any installed hook routine will be invoked after the new address information is applied.

RETURNS OK if notification hook added, or ERROR otherwise.

ERRNO S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED

SEE ALSO dhcpcLib

dhcpcEventHookDelete()

NAME *dhcpcEventHookDelete()* – remove the configuration parameters handler

SYNOPSIS

```
STATUS dhcpcEventHookDelete
(
    void * pCookie /* identifier returned by dhcpcInit() */
)
```

DESCRIPTION This routine removes the hook routine that handled changes in the configuration parameters for the lease indicated by *pCookie*. If the lease initialization specified automatic configuration of the corresponding network interface, the assigned address could change without warning after this routine is executed.

RETURNS OK if notification hook removed, or ERROR otherwise.

ERRNO S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED

SEE ALSO dhcpcLib

dhcpcInit()

NAME *dhcpcInit()* – assign network interface and setup lease request

SYNOPSIS

```
void * dhcpcInit
(
    struct ifnet * pIf, /* network device used by client */
    BOOL autoConfig /* reconfigure network device? */
)
```

DESCRIPTION This routine creates the data structures used to obtain a set of parameters with DHCP and must be called before each attempt at establishing a DHCP lease, but after the *dhcpcLibInit()* routine has initialized the global data structures. The *pIf* argument indicates the network device which will be used for transmission and reception of DHCP messages during the lifetime of the lease. If the *autoConfig* parameter is set to TRUE, any

address information obtained will automatically be applied to that interface. The specified interface must access a device capable of sending broadcast messages. Currently, only Ethernet devices and the shared-memory network drivers are supported.

The routine also uses the *autoConfig* parameter to select the default option request list for a lease. If set to FALSE, no specific lease options are requested since any configuration parameters obtained are not intended for the underlying network device. In that case, any specific options required may be added to the request list at any time before the corresponding *dhcpcBind()* call. If *autoConfig* is TRUE, this routine sets the configuration parameters to request the minimal address information (subnet mask and broadcast address) necessary for reconfiguring the network device specified by *plf*.

The internal lease identifier returned by this routine must be used in subsequent calls to the DHCP client library.

| | |
|-----------------|---|
| NOTE | This routine is called automatically during system startup if the DHCP client was used to obtain the VxWorks boot parameters. The resulting lease will always reconfigure the network boot device. Therefore, any further calls to this routine which specify the network boot device for use in obtaining additional DHCP leases must set <i>autoConfig</i> to FALSE. Otherwise, that device will be unable to maintain a stable configuration. The global variable <i>pDhcpBootCookie</i> provides access to the configuration parameters for any DHCP lease created during system startup. |
| RETURNS | Lease handle for later use, or NULL if lease setup fails. |
| ERRNO | <i>S_dhcpcLib_NOT_INITIALIZED</i> , <i>S_dhcpcLib_NO_DEVICE</i> , <i>S_dhcpcLib_BAD_OPTION</i> , <i>S_dhcpcLib_MAX_LEASES_REACHED</i> , <i>S_dhcpcLib_MEM_ERROR</i> |
| SEE ALSO | <i>dhcpcLib</i> , <i>dhcpcOptionSet()</i> , <i>dhcpcEventHookAdd()</i> |

dhcpcLibInit()

NAME *dhcpcLibInit()* – DHCP client library initialization

SYNOPSIS

```
STATUS dhcpcLibInit
(
    int serverPort,    /* port used by DHCP servers (default 67) */
    int clientPort,   /* port used by DHCP clients (default 68) */
    int maxLeases,    /* max number of simultaneous leases allowed */
    int offerTimeout, /* interval to get additional DHCP offers */
    int defaultLease, /* default value for requested lease length */
    int minLease      /* minimum accepted lease length */
)
```

| | |
|--------------------|---|
| DESCRIPTION | This routine creates and initializes the global data structures used by the DHCP client library to maintain multiple leases, up to the limit specified by the <i>maxLeases</i> parameter. Every subsequent lease attempt will collect additional DHCP offers until the interval specified by <i>offerTimeout</i> expires and will request the lease duration indicated by <i>defaultLease</i> . This routine must be called before calling any other library routines. The routine is called automatically if <code>INCLUDE_DHCP</code> is defined at the time the system is built and assigns the global lease settings to the values specified by <code>DHCPC_SPORT</code> , <code>DHCPC_CPORT</code> , <code>DHCPC_MAX_LEASES</code> , <code>DHCPC_DEFAULT_LEASE</code> , and <code>DHCPC_OFFER_TIMEOUT</code> . |
| RETURNS | OK, or ERROR if initialization fails. |
| ERRNO | <code>S_dhcpcLib_MEM_ERROR</code> |
| SEE ALSO | <code>dhcpcLib</code> |

dhcpcOptionGet()

NAME *dhcpcOptionGet()* – retrieve an option provided to a client and store in a buffer

SYNOPSIS

```
STATUS dhcpcOptionGet
(
    void * pCookie, /* identifier returned by dhcpcInit() */
    int    option, /* RFC 1533 option tag */
    int *  pLength, /* size of provided buffer and data returned */
    char * pBuf    /* location for option data */
)
```

DESCRIPTION This routine retrieves the data for the specified *option*, if present for the lease indicated by *pCookie*. The data is stored in the provided buffer, whose length must be specified. If the *option* is found, the amount of data available is stored in the location referenced by the *pLength* parameter. The option is not available if the DHCP client is not in the bound state or if the server did not provide it. After returning, the provided buffer may contain IP addresses stored in network byte order. All other numeric values are stored in host byte order. See RFC 1533 for specific details on the data retrieved.

| | |
|-----------------|---|
| RETURNS | OK if option available, or ERROR otherwise. |
| ERRNO | <code>S_dhcpcLib_BAD_COOKIE</code> , <code>S_dhcpcLib_NOT_INITIALIZED</code> , <code>S_dhcpcLib_NOT_BOUNDED</code> , <code>S_dhcpcLib_OPTION_NOT_PRESENT</code> |
| SEE ALSO | <code>dhcpcLib</code> , <i>dhcpcOptionSet()</i> |

dhcpcOptionSet()

NAME *dhcpcOptionSet()* – add an option to the option request list

SYNOPSIS

```
STATUS dhcpcOptionSet
(
    void * pCookie, /* identifier returned by dhcpcInit() */
    int    option, /* RFC 1533 tag of desired option */
    long   value,  /* numeric value for option */
    long   length, /* length of data (if any) or 0 if unused */
    char * pData   /* option data, or NULL if none */
)
```

DESCRIPTION This routine sets most client-to-server transmission options for the lease indicated by the *pCookie* parameter. The *option* parameter specifies an option tag as defined in RFC 1533 and the updates published in the Internet Draft of November 1996. For a listing of defined aliases for the known option tags, see **dhcpc/dhpc.h**. This routine cannot set the options associated with the following tags:

```
_DHCP_PAD_TAG
_DHCP_OPT_OVERLOAD_TAG
_DHCP_MSGTYPE_TAG
_DHCP_SERVER_ID_TAG
_DHCP_REQ_LIST_TAG
_DHCP_MAXMSGSIZE_TAG
_DHCP_END_TAG
```

Most options only require specification of the appropriate tag in the *option* parameter. In those cases, the *dhcpcOptionSet()* call adds the specified option tag to the option request list, if possible. However, some options require additional information. The tags for these options are:

```
_DHCP_VENDOR_SPEC_TAG
_DHCP_REQUEST_IPADDR_TAG
_DHCP_LEASE_TIME_TAG
_DHCP_ERRMSG_TAG
_DHCP_CLASS_ID_TAG
_DHCP_CLIENT_ID_TAG
```

The `_DHCP_LEASE_TIME_TAG` and `_DHCP_CLIENT_ID_TAG` options each use the *value* parameter. For `_DHCP_LEASE_TIME_TAG`, *value* specifies the desired lease length. For `_DHCP_CLIENT_ID_TAG`, *value* specifies the type for a type/value pair. No other options use this parameter.

The `_DHCP_VENDOR_SPEC_TAG`, `_DHCP_CLASS_ID_TAG` and `_DHCP_CLIENT_ID_TAG` tags each require a value for the *length* parameter to specify the number of bytes of data provided. No other options use this parameter.

The *pData* parameter is relevant to the following option tags:

`_DHCP_VENDOR_SPEC_TAG`

The *pData* parameter references a list of *length* bytes of options in the format specified by RFC 1533.

`_DHCP_REQUEST_IPADDR_TAG`

The *pData* parameter indicates the string representation of the desired Internet address for the client in dot notation.

`_DHCP_ERRMSG_TAG`

The *pData* parameter indicates the error message to send to the server when releasing the current IP address. That location must be valid until the release is completed, since the message is not copied or stored in any way.

`_DHCP_CLASS_ID_TAG`

The *pData* parameter references *length* bytes used as the value for the vendor class identifier.

`_DHCP_CLIENT_ID_TAG`

The *pData* parameter references *length* bytes used as the value of a type/value pair.

-

The *pData* parameter is not used by any other options.

| | |
|-----------------|--|
| NOTE | With the exception of the <code>_DHCP_ERRMSG_TAG</code> option, the DHCP specification forbids changing options after a lease has been established. Therefore, this routine should not be used after the <code>dhcpcBind()</code> call. Changing any option other than the error message at that point could have unpredictable results. |
| RETURNS | OK if the option was set successfully, or ERROR if the option is invalid or storage failed. |
| ERRNO | <code>S_dhcpcLib_BAD_OPTION</code> , <code>S_dhcpcLib_OPTION_NOT_STORED</code> |
| SEE ALSO | <code>dhcpcLib</code> |

dhcpcParamsGet()

| | |
|--------------------|---|
| NAME | <i>dhcpcParamsGet()</i> – retrieve current configuration parameters |
| SYNOPSIS | <pre>STATUS dhcpcParamsGet (void * pCookie, /* identifier returned by dhcpcInit() */ struct dhcp_param * pParamList /* requested parameters */)</pre> |
| DESCRIPTION | <p>This routine copies the current configuration parameters for the lease specified by the <i>pCookie</i> argument to the user-supplied structure. That structure, defined in <i>dhcpc/dhcpc.h</i>, should contain non-NULL pointers to indicate the parameters of interest. All other values within the structure must be set to 0 before calling the routine. The requested information is only retrieved if the specified lease is in the bound state and knows that its parameters are good.</p> <p>Many of the parameters within the user-supplied structure use one of the following secondary data types: <i>struct in_addr</i>, <i>struct u_shorts</i>, and <i>struct vendor_list</i>. Each of those structures accepts a length designation and a data pointer. For the first two data types, the num member indicates the size of the buffer in terms of the number of underlying elements. For example, the STATIC_ROUTE option returns one or more IP address pairs. So, setting the num member to 2 in the <i>static_route</i> entry would indicate that the corresponding buffer contained 16 bytes. By contrast, the len member in the <i>struct vendor_list</i> data type consists of the buffer size, in bytes. See RFC 1533 for specific details on the types of data for each option.</p> <p>On return, each of the length designators are set to indicate the amount of data returned. For instance, the num member in the <i>static_route</i> entry could be set to 1 to indicate that only one IP address pair of 8 bytes was available.</p> |
| RETURNS | OK if in bound state, or ERROR otherwise. |
| ERRNO | <i>S_dhcpcLib_BAD_COOKIE</i> , <i>S_dhcpcLib_NOT_INITIALIZED</i> , <i>S_dhcpcLib_NOT_BOUND</i> |
| SEE ALSO | <i>dhcpcLib</i> |

dhcpcParamsShow()

| | |
|--------------------|--|
| NAME | <i>dhcpcParamsShow()</i> – display current lease parameters |
| SYNOPSIS | <pre>STATUS dhcpcParamsShow (void * pCookie /* identifier returned by dhcpcInit() */)</pre> |
| DESCRIPTION | This routine prints all lease parameters for the lease identified by <i>pCookie</i> . It has no effect if the indicated lease is not currently active. |
| RETURNS | OK, or ERROR if lease identifier unknown. |
| ERRNO | S_dhcpcLib_BAD_COOKIE |
| SEE ALSO | <i>dhcpcShow</i> |

dhcpcRelease()

| | |
|--------------------|---|
| NAME | <i>dhcpcRelease()</i> – relinquish specified lease |
| SYNOPSIS | <pre>STATUS dhcpcRelease (void * pCookie /* identifier returned by dhcpcInit() */)</pre> |
| DESCRIPTION | This routine schedules the lease identified by the <i>pCookie</i> parameter for immediate release, regardless of time remaining, and removes all the associated data structures. After the release completes, a new call to <i>dhcpcInit()</i> is required before attempting another lease. |
| NOTE | This routine will disable the underlying network interface if automatic configuration was requested. This may occur without warning if no event hook is installed. |
| RETURNS | OK if release scheduled, or ERROR otherwise. |
| ERRNO | S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED |
| SEE ALSO | <i>dhcpcLib</i> |

dhcpcServerGet()

| | |
|--------------------|--|
| NAME | <i>dhcpcServerGet()</i> – retrieve the current DHCP server |
| SYNOPSIS | <pre>STATUS dhcpcServerGet (void * pCookie, /* identifier returned by dhcpcInit() */ struct in_addr * pServerAddr /* location for address of server */)</pre> |
| DESCRIPTION | This routine returns the DHCP server that supplied the configuration parameters for the lease specified by the <i>pCookie</i> argument. This information is available only if the lease is in the bound state. |
| RETURNS | OK if in bound state and server available, or ERROR otherwise. |
| ERRNO | S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED, S_dhcpcLib_NOT_BOUND |
| SEE ALSO | <i>dhcpcLib</i> |

dhcpcServerShow()

| | |
|--------------------|--|
| NAME | <i>dhcpcServerShow()</i> – display current DHCP server |
| SYNOPSIS | <pre>STATUS dhcpcServerShow (void * pCookie /* identifier returned by dhcpcInit() */)</pre> |
| DESCRIPTION | This routine prints the IP address of the DHCP server that provided the parameters for the lease identified by <i>pCookie</i> . It has no effect if the indicated lease is not currently active. |
| RETURNS | OK, or ERROR if lease identifier unknown. |
| ERRNO | S_dhcpcLib_BAD_COOKIE |
| SEE ALSO | <i>dhcpcShow</i> |

dhcpcShowInit()

| | |
|--------------------|---|
| NAME | <i>dhcpcShowInit()</i> – initialize the DHCP show facility |
| SYNOPSIS | <code>void dhcpcShowInit (void)</code> |
| DESCRIPTION | This routine links the DHCP show facility into the VxWorks system image. It is called from <code>usrNetwork.c</code> automatically if <code>INCLUDE_DHCP</code> and <code>INCLUDE_NET_SHOW</code> are defined at the time the image is constructed. |
| SEE ALSO | <code>dhcpcShow</code> |

dhcpcShutdown()

| | |
|--------------------|--|
| NAME | <i>dhcpcShutdown()</i> – disable DHCP client library |
| SYNOPSIS | <code>STATUS dhcpcShutdown (void)</code> |
| DESCRIPTION | This routine schedules the lease monitor task to clean up memory and exit, after releasing all currently active leases. The network boot device will be disabled if the DHCP client was used to obtain the VxWorks boot parameters and the resulting lease is still active. Any other interfaces using the addressing information from leases set for automatic configuration will also be disabled. Notification of a disabled interface will not occur unless an event hook has been installed. After the processing started by this request completes, the DHCP client library is unavailable until restarted with the <i>dhcpcLibInit()</i> routine. |
| RETURNS | OK if shutdown scheduled, or ERROR otherwise. |
| ERRNO | <code>S_dhcpcLib_NOT_INITIALIZED</code> |
| SEE ALSO | <code>dhcpcLib</code> |

dhcpcTimerGet()

| | |
|--------------------|---|
| NAME | <i>dhcpcTimerGet()</i> – retrieve current lease timers |
| SYNOPSIS | <pre>STATUS dhcpcTimerGet (void * pCookie, /* identifier returned by dhcpcInit() */ int * pT1, /* time until lease renewal */ int * pT2 /* time until lease rebinding */)</pre> |
| DESCRIPTION | This routine returns the number of clock ticks remaining on the timers governing the DHCP lease specified by the <i>pCookie</i> argument. This information is only available if the lease is in the bound state. Therefore, this routine will return ERROR if a BOOTP reply was accepted. |
| RETURNS | OK if in bound state and values available, or ERROR otherwise. |
| ERRNO | S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED, S_dhcpcLib_NOT_BOUNDED, S_dhcpcLib_OPTION_NOT_PRESENT, S_dhcpcLib_TIMER_ERROR |
| SEE ALSO | <i>dhcpcLib</i> |

dhcpcTimersShow()

| | |
|--------------------|---|
| NAME | <i>dhcpcTimersShow()</i> – display current lease timers |
| SYNOPSIS | <pre>STATUS dhcpcTimersShow (void * pCookie /* identifier returned by dhcpcInit() */)</pre> |
| DESCRIPTION | This routine prints the time remaining with each of the DHCP lease timers for the lease identified by <i>pCookie</i> . It has no effect if the indicated lease is not currently active. |
| RETURNS | OK if show routine completes, or ERROR otherwise. |
| ERRNO | S_dhcpcLib_BAD_COOKIE |
| SEE ALSO | <i>dhcpcShow</i> |

dhcpcVerify()

| | |
|--------------------|---|
| NAME | <i>dhcpcVerify()</i> – renew an established lease |
| SYNOPSIS | <pre>STATUS dhcpcVerify (void * pCookie /* identifier returned by dhcpcInit() */)</pre> |
| DESCRIPTION | This routine schedules the lease identified by the <i>pCookie</i> parameter for immediate renewal according to the process described in RFC 1541. If the renewal is unsuccessful, the lease negotiation process restarts. The routine is valid as long as the lease is currently active. The routine is also called automatically in response to a <i>dhcpcBind()</i> call for an existing lease. |
| NOTE | This routine will disable the underlying network interface if the verification fails and automatic configuration was requested. This may occur without warning if no event hook is installed. |
| RETURNS | OK if verification scheduled, or ERROR otherwise. |
| ERRNO | S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED, S_dhcpcLib_NOT_BOUND |
| SEE ALSO | dhcpcLib |

dhcpsAddressHookAdd()

| | |
|--------------------|---|
| NAME | <i>dhcpsAddressHookAdd()</i> – assign a permanent address storage hook for the server |
| SYNOPSIS | <pre>STATUS dhcpsAddressHookAdd (FUNCPTR pCacheHookRtn /* routine to store/retrieve lease entries */)</pre> |
| DESCRIPTION | This routine allows the server to access some form of permanent storage to preserve additional address entries across restarts. This routine is not required, but leases using unsaved addresses are not renewed. The only argument provided is the name of a function with the following interface: <pre>STATUS dhcpsAddressStorageHook (int op, char *name, char *start, char *end, char *params);</pre> |

The first parameter of this storage routine specifies one of the following operations:

```
DHCPS_STORAGE_START
DHCPS_STORAGE_READ
DHCPS_STORAGE_WRITE
DHCPS_STORAGE_STOP
```

In response to a START, the storage routine should prepare to return data or overwrite data provided by earlier WRITE operations. For a WRITE, the storage routine must save the contents of the four buffers to permanent storage. Those buffers contain the NULL-terminated strings received by the *dhcpsLeaseEntryAdd()* routine. For a READ, the storage routine should copy previously stored data (as NULL-terminated strings) into the provided buffers in the order received by earlier WRITE operations. For a STOP, the storage routine should do any necessary cleanup. After a STOP, the storage routine should return an ERROR for all operations except START.

The storage routine should return OK if successful, ERROR otherwise.

Note that, unlike the lease storage routine, there is no CLEAR operation.

Before the server is initialized, VxWorks calls this routine automatically passing in the function named in DHCPS_ADDRESS_HOOK.

RETURNS OK, or ERROR if function pointer is NULL.

ERRNO N/A

SEE ALSO *dhcpsLib*

dhcpsInit()

NAME *dhcpsInit()* – set up the DHCP server parameters and data structures

SYNOPSIS

```
STATUS dhcpsInit
(
    struct ifnet * * ppIf,          /* network devices used by server */
    int numDev,                  /* number of devices */
    DHCPS_LEASE_DESC * pLeasePool, /* table of lease data */
    int poolSize,                /* size of data table */
    DHCPS_RELAY_DESC * pRelayTbl, /* table of relay agent data */
    int relaySize,                /* size of relay agent table */
    DHCP_TARGET_DESC * pTargetTbl, /* table of receiving DHCP servers */
    int targetSize
)
```

DESCRIPTION This routine creates the necessary data structures, builds the server address pool, retrieves any lease or address information from permanent storage through the user-provided hooks, and initializes the network interfaces for monitoring. It is called at system startup if `INCLUDE_DHCP` is defined at the time the VxWorks image is built.

RETURNS OK, or ERROR if could not initialize.

SEE ALSO `dhcpsLib`

dhcpsLeaseEntryAdd()

NAME `dhcpsLeaseEntryAdd()` – add another entry to the address pool

SYNOPSIS

```
STATUS dhcpsLeaseEntryAdd
(
    char * pName,      /* name of lease entry */
    char * pStartIp,  /* first IP address to assign */
    char * pEndIp,    /* last IP address in assignment range */
    char * pParams    /* formatted string of lease parameters */
)
```

DESCRIPTION This routine allows the user to add new entries to the address pool without rebuilding the VxWorks image. The routine requires a unique entry name of up to eight characters, starting and ending IP addresses, and a colon-separated list of parameters. Possible values for the parameters are listed in the reference entry for `dhcpsLib`. The parameters also determine the type of lease, which the server uses to determine priority when assigning lease addresses. For examples of the possible lease types, see the reference entry for `dhcpsLib`.

RETURNS OK if entry read successfully, or ERROR otherwise.

ERRNO N/A

SEE ALSO `dhcpsLib`

dhcpsLeaseHookAdd()

NAME *dhcpsLeaseHookAdd()* – assign a permanent lease storage hook for the server

SYNOPSIS

```
STATUS dhcpsLeaseHookAdd
(
    FUNCPTR pCacheHookRtn /* routine to store/retrieve lease records */
)
```

DESCRIPTION This routine allows the server to access some form of permanent storage that it can use to store current lease information across restarts. The only argument to *dhcpsLeaseHookAdd()* is a pointer to a storage routine with the following interface:

```
STATUS dhcpsStorageHook (int op, char *buffer, int datalen);
```

The first parameter of the storage routine specifies one of the following operations:

```
DHCPS_STORAGE_START
DHCPS_STORAGE_READ
DHCPS_STORAGE_WRITE
DHCPS_STORAGE_STOP
DHCPS_STORAGE_CLEAR
```

In response to **START**, the storage routine should prepare to return data or overwrite data provided by earlier **WRITES**. For a **WRITE** the storage routine must save the contents of the buffer to permanent storage. For a **READ**, the storage routine should copy data previously stored into the provided buffer as a **NULL**-terminated string in **FIFO** order. For a **CLEAR**, the storage routine should discard currently stored data. After a **CLEAR**, the **READ** operation must return **ERROR** until additional data is stored. For a **STOP**, the storage routine must handle cleanup. After a **STOP**, calls to the storage routine must return error until a **START** is received. Each of these operations must return **OK** if successful, or **ERROR** otherwise.

Before the server is initialized, **VxWorks** automatically calls *dhcpsLeaseHookAdd()*, passing in the routine name defined by **DHCPS_LEASE_HOOK**.

RETURNS **OK**, or **ERROR** if routine is **NULL**.

ERRNO **N/A**

SEE ALSO **dhcpsLib**

difftime()

difftime()

| | |
|----------------------|---|
| NAME | <i>difftime()</i> – compute the difference between two calendar times (ANSI) |
| SYNOPSIS | <pre>double difftime (time_t time1, /* later time, in seconds */ time_t time0 /* earlier time, in seconds */)</pre> |
| DESCRIPTION | This routine computes the difference between two calendar times: <i>time1</i> - <i>time0</i> . |
| INCLUDE FILES | time.h |
| RETURNS | The time difference in seconds, expressed as a double. |
| SEE ALSO | ansiTime |

diskFormat()

| | |
|--------------------|---|
| NAME | <i>diskFormat()</i> – format a disk |
| SYNOPSIS | <pre>STATUS diskFormat (char * devName /* name of the device to initialize */)</pre> |
| DESCRIPTION | <p>This command formats a disk and creates a file system on it. The device must already have been created by the device driver and initialized for use with a particular file system, via <i>dosFsDevInit()</i> or <i>rt11FsDevInit()</i>.</p> <p>This command calls <i>ioctl()</i> to perform the FIODISKFORMAT function.</p> |
| EXAMPLE | <pre>-> diskFormat "/fd0/"</pre> |
| RETURNS | OK, or ERROR if the device cannot be opened or formatted. |
| SEE ALSO | usrLib, dosFsLib, rt11FsLib , <i>VxWorks Programmer's Guide: Target Shell</i> |

diskInit()

| | |
|--------------------|---|
| NAME | <i>diskInit()</i> – initialize a file system on a block device |
| SYNOPSIS | <pre>STATUS diskInit (char * devName /* name of the device to initialize */)</pre> |
| DESCRIPTION | This command creates a new, blank file system on a block device. The device must already have been created by the device driver and initialized for use with a particular file system, via <i>dosFsDevInit()</i> or <i>rt11FsDevInit()</i> . This command calls <i>ioctl()</i> to perform the FIODISKINIT function. |
| EXAMPLE | <pre>-> diskInit "/fd0/"</pre> |
| RETURNS | OK, or ERROR if the device cannot be opened or initialized. |
| SEE ALSO | <i>usrLib</i> , <i>dosFsLib</i> , <i>rt11FsLib</i> , <i>VxWorks Programmer's Guide: Target Shell</i> |

div()

| | |
|----------------------|--|
| NAME | <i>div()</i> – compute a quotient and remainder (ANSI) |
| SYNOPSIS | <pre>div_t div (int numer, /* numerator */ int denom /* denominator */)</pre> |
| DESCRIPTION | This routine computes the quotient and remainder of <i>numer/denom</i> . If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, quot * <i>denom</i> + reme equals <i>numer</i> . This routine is not reentrant. For a reentrant version, see <i>div_r()</i> . |
| INCLUDE FILES | <i>stdlib.h</i> |
| RETURNS | A structure of type <i>div_t</i> , containing both the quotient and the remainder. |
| SEE ALSO | <i>ansiStdlib</i> |

div_r()

NAME *div_r()* – compute a quotient and remainder (reentrant)

SYNOPSIS

```
void div_r
(
    int    numer,      /* numerator */
    int    denom,      /* denominator */
    div_t * divStructPtr /* div_t structure */
)
```

DESCRIPTION This routine computes the quotient and remainder of *numer/denom*. The quotient and remainder are stored in the **div_t** structure pointed to by *divStructPtr*.

This routine is the reentrant version of *div()*.

INCLUDE FILES `stdlib.h`

RETURNS N/A

SEE ALSO `ansiStdlib`

dosFsConfigGet()

NAME *dosFsConfigGet()* – obtain dosFs volume configuration values

SYNOPSIS

```
STATUS dosFsConfigGet
(
    DOS_VOL_DESC *  vdptr, /* ptr to volume descriptor */
    DOS_VOL_CONFIG * pConfig /* ptr to config structure to fill */
)
```

DESCRIPTION This routine obtains the current configuration values for a dosFs disk volume. The data is obtained from the dosFs volume descriptor specified by *vdptr*. No physical I/O to the device takes place.

The configuration data is placed into a **DOS_VOL_CONFIG** structure, whose address is *pConfig*. This structure must be allocated before calling *dosFsConfigGet()*.

One use for this routine is to obtain the configuration data from a known good disk, to be used to initialize a new disk (using *dosFsDevInit()*).

The volume is not locked while the data is being read from the volume descriptor, so it is conceivable that another task may modify the configuration information while this routine is executing.

RETURNS OK or ERROR.

SEE ALSO *dosFsLib*

dosFsConfigInit()

NAME *dosFsConfigInit()* – initialize dosFs volume configuration structure

SYNOPSIS

```
STATUS dosFsConfigInit
(
    DOS_VOL_CONFIG * pConfig,    /* pointer to volume config structure */
    char           mediaByte,    /* media descriptor byte */
    UINT8          secPerClust,  /* sectors per cluster */
    short          nResrvd,      /* number of reserved sectors */
    char           nFats,        /* number of FAT copies */
    UINT16         secPerFat,    /* number of sectors per FAT copy */
    short          maxRootEnts,  /* max number of entries in root dir */
    UINT           nHidden,      /* number of hidden sectors */
    UINT           options       /* volume options */
)
```

DESCRIPTION This routine initializes a dosFs volume configuration structure (DOS_VOL_CONFIG). This structure is used by the *dosFsDevInit()* routine to specify the file system configuration for the disk.

The DOS_VOL_CONFIG structure must have been allocated prior to calling this routine. Its address is specified by *pConfig*. The specified configuration variables are placed into their respective fields in the structure.

This routine is provided only to allow convenient initialization of the DOS_VOL_CONFIG structure (particularly from the VxWorks shell). A structure which is properly initialized by other means may be used equally well by *dosFsDevInit()*.

RETURNS OK, or ERROR if there is an invalid parameter or *pConfig* is NULL.

SEE ALSO *dosFsLib*, *dosFsDevInit()*

dosFsConfigShow()

dosFsConfigShow()**NAME** *dosFsConfigShow()* – display dosFs volume configuration data**SYNOPSIS**

```
STATUS dosFsConfigShow
(
    char * devName /* name of device */
)
```

DESCRIPTION This routine obtains the dosFs volume configuration for the named device, formats the data, and displays it on the standard output. The information which is displayed is that which is contained in a `DOS_VOL_CONFIG` structure, along with other configuration values (for example, from the `BLK_DEV` structure which describes the device).

If no device name is specified, the current default device is described.

RETURNS OK or ERROR.**SEE ALSO** `dosFsLib`

dosFsDateSet()**NAME** *dosFsDateSet()* – set the dosFs file system date**SYNOPSIS**

```
STATUS dosFsDateSet
(
    int year, /* year (1980...2099) */
    int month, /* month (1...12) */
    int day /* day (1...31) */
)
```

DESCRIPTION This routine sets the date for the dosFs file system, which remains in effect until changed. All files created or modified are assigned this date in their directory entries.**NOTE** No automatic incrementing of the date is performed; each new date must be set with a call to this routine.**RETURNS** OK, or ERROR if the date is invalid.**SEE ALSO** `dosFsLib`, `dosFsTimeSet()`, `dosFsDateTimeInstall()`

dosFsDateTimeInstall()

- NAME** *dosFsDateTimeInstall()* – install a user-supplied date/time function
- SYNOPSIS**
- ```
void dosFsDateTimeInstall
(
 FUNCPTR pDateTimeFunc /* pointer to user-supplied function */
)
```
- DESCRIPTION**
- This routine installs a user-supplied function to provide the current date and time. Once such a function is installed, **dosFsLib** will call it when necessary to obtain the date and time. Otherwise, the date and time most recently set by *dosFsDateSet()* and *dosFsTimeSet()* are used.
- The user-supplied routine must take exactly one input parameter, the address of a **DOS\_DATE\_TIME** structure (defined in **dosFsLib.h**). The user routine should update the necessary fields in this structure and then return. Any fields which are not changed by the user routine will retain their previous value.
- RETURNS** N/A
- SEE ALSO** **dosFsLib**

---

## *dosFsDevInit()*

- NAME** *dosFsDevInit()* – associate a block device with dosFs file system functions
- SYNOPSIS**
- ```
DOS_VOL_DESC *dosFsDevInit
(
    char *          devName, /* device name */
    BLK_DEV *      pBlkDev, /* pointer to block device struct */
    DOS_VOL_CONFIG * pConfig /* pointer to volume config data */
)
```
- DESCRIPTION**
- This routine takes a block device structure (**BLK_DEV**) created by a device driver and defines it as a dosFs volume. As a result, when high-level I/O operations (e.g., *open()*, *write()*) are performed on the device, the calls will be routed through **dosFsLib**. The *pBlkDev* parameter is the address of the **BLK_DEV** structure which describes this device.
- This routine associates the name *devName* with the device and installs it in the VxWorks I/O system's device table. The driver number used when the device is added to the table

dosFsDevInitOptionsSet()

is that which was assigned to the dosFs library during *dosFsInit()*. (The driver number is placed in the global variable **dosFsDrvNum**.)

The **BLK_DEV** structure contains configuration data describing the device and the addresses of five routines which will be called to read sectors, write sectors, reset the device, check device status, and perform other control functions (*ioctl()*). These routines will not be called until they are required by subsequent I/O operations.

The *pConfig* parameter is the address of a **DOS_VOL_CONFIG** structure. This structure must have been previously initialized with the specific dosFs configuration data to be used for this volume. This structure may be easily initialized using *dosFsConfigInit()*.

If the device being initialized already has a valid dosFs (MS-DOS) file system on it, the *pConfig* parameter may be NULL. In this case, the volume will be mounted and the configuration data will be read from the boot sector of the disk. (If *pConfig* is NULL, both change-no-warn and auto-sync options are initially disabled. These can be enabled using the *dosFsVolOptionsSet()* routine.)

This routine allocates and initializes a volume descriptor (**DOS_VOL_DESC**) for the device. It returns a pointer to **DOS_VOL_DESC**.

RETURNS A pointer to the volume descriptor **DOS_VOL_DESC**, or NULL if there is an error.

SEE ALSO *dosFsLib*, *dosFsMkfs()*

dosFsDevInitOptionsSet()

NAME *dosFsDevInitOptionsSet()* – specify volume options for *dosFsDevInit()*

SYNOPSIS

```
STATUS dosFsDevInitOptionsSet
(
    UINT options /* options for future dosFsDevInit() calls */
)
```

DESCRIPTION This routine allows volume options to be set that will be enabled by subsequent calls to *dosFsDevInit()* that do not explicitly supply configuration information in a **DOS_VOL_CONFIG** structure. This is normally done when mounting a disk which has already been initialized with file system data. The value of *options* will be used for all volumes that are initialized by *dosFsDevInit()*, unless a specific configuration is given.

The only volume options which may be specified in this call are those which are not tied to the actual data on the disk. Specifically, you may not specify the long file name option in this call; if a disk using that option is mounted, that will be automatically detected. If you specify such an unsettable option during this call it will be ignored; all valid option bits will still be accepted and applied during subsequent *dosFsDevInit()* calls.

For example, to use *dosFsDevInit()* to initialize a volume with the auto-sync and filesystem export options, do the following:

```
status = dosFsDevInitOptionsSet (DOS_OPT_AUTOSYNC | DOS_OPT_EXPORT);
if (status != OK)
    return (ERROR);
vdptr = dosFsDevInit ("DEV1:", pBlkDev, NULL);
/* note NULL pointer for DOS_VOL_CONFIG */
```

RETURNS OK, or ERROR if *options* is invalid.

SEE ALSO *dosFsLib*, *dosFsDevInit()*, *dosFsVolOptionsSet()*

dosFsInit()

NAME *dosFsInit()* – prepare to use the dosFs library

SYNOPSIS

```
STATUS dosFsInit
(
    int maxFiles /* max no. of simultaneously open files */
)
```

DESCRIPTION This routine initializes the dosFs library. It must be called exactly once, before any other routine in the library. The argument specifies the number of dosFs files that may be open at once. This routine installs **dosFsLib** as a driver in the I/O system driver table, allocates and sets up the necessary memory structures, and initializes semaphores. The driver number assigned to **dosFsLib** is placed in the global variable **dosFsDrvNum**.

This initialization is enabled when the configuration macro **INCLUDE_DOSFS** is defined; *dosFsInit()* is then called from the root task, *usrRoot()*, in *usrConfig.c*.

RETURNS OK or ERROR.

SEE ALSO *dosFsLib*

dosFsMkfs()

dosFsMkfs()

NAME *dosFsMkfs()* – initialize a device and create a dosFs file system

SYNOPSIS

```
DOS_VOL_DESC *dosFsMkfs
(
    char *    volName, /* volume name to use */
    BLK_DEV * pBlkDev /* pointer to block device struct */
)
```

DESCRIPTION This routine provides a quick method of creating a dosFs file system on a device. It is used instead of the two-step procedure of calling *dosFsDevInit()* followed by an *ioctl()* call with an FIODISKINIT function code.

This call uses default values for various dosFs configuration parameters (i.e., those found in the volume configuration structure, *DOS_VOL_CONFIG*). The values used are:

- 2 sectors per cluster (see below)
- 1 reserved sector
- 2 FAT copies
- 112 root directory entries
- 0xF0 media byte value
- 0 hidden sectors

The volume options (auto-sync mode, change-no-warn mode, and long filenames) that are enabled by this routine can be set in advance using *dosFsMkfsOptionsSet()*. By default, none of these options is enabled for disks initialized by *dosFsMkfs()*.

If initializing a large disk, it is quite possible that the entire disk area cannot be described by the maximum 64K clusters if only two sectors are contained in each cluster. In such a situation, *dosFsMkfs()* will automatically increase the number of sectors per cluster to a number which will allow the entire disk area to be described in 64K clusters.

The number of sectors per FAT copy is set to the minimum number of sectors which will contain sufficient FAT entries for the entire block device.

RETURNS A pointer to a dosFs volume descriptor, or NULL if there is an error.

ERRNO *S_dosFsLib_INVALID_PARAMETER*

SEE ALSO *dosFsLib*, *dosFsDevInit()*

dosFsMkfsOptionsSet()

- NAME** *dosFsMkfsOptionsSet()* – specify volume options for *dosFsMkfs()*
- SYNOPSIS**
- ```
STATUS dosFsMkfsOptionsSet
(
 UINT options /* options for future dosFsMkfs() calls */
)
```
- DESCRIPTION**
- This routine allows volume options to be set that will be enabled by subsequent calls to *dosFsMkfs()*. The value of *options* will be used for all volumes initialized by *dosFsMkfs()*. For example, to use *dosFsMkfs()* to initialize a volume with the auto-sync and long filename options, do the following:
- ```
status = dosFsMkfsOptionsSet (DOS_OPT_AUTOSYNC | DOS_OPT_LONGNAMES);
if (status != OK)
    return (ERROR);
vdptr = dosFsMkfs ("DEV1:", pBlkDev);
```
- RETURNS** OK, or ERROR if *options* is invalid.
- SEE ALSO** *dosFsLib*, *dosFsMkfs()*, *dosFsVolOptionsSet()*

dosFsModeChange()

- NAME** *dosFsModeChange()* – modify the mode of a dosFs volume
- SYNOPSIS**
- ```
void dosFsModeChange
(
 DOS_VOL_DESC * vdptr, /* pointer to volume descriptor */
 int newMode /* O_RDONLY/O_WRONLY/O_RDWR (both) */
)
```
- DESCRIPTION**
- This routine sets the volume's mode to *newMode*. The mode is actually kept in "bd\_mode" fields of the the **BLK\_DEV** structure, so that it may also be used by the device driver. Changing that field directly has the same result as calling this routine. The mode field should be updated whenever the read and write capabilities are determined, usually after a ready change. See the manual entry for *dosFsReadyChange()*.
- The driver's device initialization routine should initially set the mode field to **O\_RDWR** (i.e., both **O\_RDONLY** and **O\_WRONLY**).

**RETURNS** N/A

**SEE ALSO** `dosFsLib`, `dosFsReadyChange()`

---

## *dosFsReadyChange()*

**NAME** `dosFsReadyChange()` – notify dosFs of a change in ready status

**SYNOPSIS**

```
void dosFsReadyChange
(
 DOS_VOL_DESC * vdptr /* pointer to volume descriptor */
)
```

**DESCRIPTION** This routine sets the volume descriptor's state to `DOS_VD_READY_CHANGED`. It should be called whenever a driver senses that a device has come on-line or gone off-line (e.g., a disk has been inserted or removed).

After this routine has been called, the next attempt to use the volume will result in an attempted remount.

This routine may also be invoked by calling `ioctl()` with `FIODISKCHANGE`.

Setting the `bd_readyChanged` field to `TRUE` in the `BLK_DEV` structure that describes this device will have the same result as calling this routine.

**RETURNS** N/A

**SEE ALSO** `dosFsLib`

---

## *dosFsTimeSet()*

**NAME** `dosFsTimeSet()` – set the dosFs file system time

**SYNOPSIS**

```
STATUS dosFsTimeSet
(
 int hour, /* 0 to 23 */
 int minute, /* 0 to 59 */
 int second /* 0 to 59 */
)
```

- DESCRIPTION** This routine sets the time for the dosFs file system, which remains in effect until changed. All files created or modified are assigned this time in their directory entries.
- NOTE** No automatic incrementing of the time is performed; each new time must be set with a call to this routine.
- RETURNS** OK, or ERROR if the time is invalid.
- SEE ALSO** *dosFsLib*, *dosFsDateSet()*, *dosFsDateTimeInstall()*

---

## *dosFsVolOptionsGet()*

- NAME** *dosFsVolOptionsGet()* – get current dosFs volume options
- SYNOPSIS**
- ```
STATUS dosFsVolOptionsGet
(
    DOS_VOL_DESC * vdptr, /* ptr to volume descriptor */
    UINT *      pOptions /* where to put current options value */
)
```
- DESCRIPTION** This routine obtains the current options for a specified dosFs volume and stores them in the field pointed to by *pOptions*.
- RETURNS** OK, always.
- SEE ALSO** *dosFsLib*, *dosFsVolOptionsSet()*

dosFsVolOptionsSet()

- NAME** *dosFsVolOptionsSet()* – set dosFs volume options
- SYNOPSIS**
- ```
STATUS dosFsVolOptionsSet
(
 DOS_VOL_DESC * vdptr, /* ptr to volume descriptor */
 UINT options /* new options for volume */
)
```
- DESCRIPTION** This routine sets the volume options for an already-initialized dosFs device. Only the following options can be changed (enabled or disabled) dynamically:

***dosFsVolUnmount()***

DOS\_OPT\_CHANGENOWARN (0x1)

DOS\_OPT\_AUTOSYNC (0x2)

The **DOS\_OPT\_CHANGENOWARN** option may be enabled only for removable volumes (i.e., the **bd\_removable** field in the **BLK\_DEV** structure for the device must be set to **TRUE**). If specified for a non-removable volume, it is ignored. When successfully set, the **DOS\_OPT\_CHANGENOWARN** option also enables the **DOS\_OPT\_AUTOSYNC** option.

It is recommended that the current volume options be obtained by calling *dosFsVolOptionsGet()*, the desired option bits modified, and then the options set using *dosFsVolOptionsSet()*.

**RETURNS** OK, or **ERROR** if *options* is invalid or an attempt is made to change an option that is not dynamically changeable.

**SEE ALSO** *dosFsLib*, *dosFsDevInitOptionsSet()*, *dosFsMkfsOptionsSet()*, *dosFsVolOptionsGet()*

***dosFsVolUnmount()***

**NAME** *dosFsVolUnmount()* – unmount a dosFs volume

**SYNOPSIS**

```
STATUS dosFsVolUnmount
(
 DOS_VOL_DESC * vdptr /* pointer to volume descriptor */
)
```

**DESCRIPTION** This routine is called when I/O operations on a volume are to be discontinued. This is the preferred action prior to changing a removable disk.

All buffered data for the volume is written to the device (if possible, with no error returned if data cannot be written), any open file descriptors are marked as obsolete, and the volume is marked as not currently mounted. When a subsequent I/O operation is initiated on the disk (e.g., during the next *open()*), the volume will be remounted automatically.

Once file descriptors have been marked as obsolete, any attempt to use them for file operations will return an error. (An obsolete file descriptor may be freed by using *close()*. The call to *close()* will return an error, but the descriptor will in fact be freed.) File descriptors obtained by opening the entire volume (in raw mode) are not marked as obsolete.

This routine may also be invoked by calling *ioctl()* with the **FIOUNMOUNT** function code.

This routine must not be called from interrupt level.



**RETURNS** OK, or ERROR if the volume was not mounted.

**SEE ALSO** `dosFsLib`, `dosFsReadyChange()`

---

## *dummyCallback()*

**NAME** `dummyCallback()` – dummy callback routine

**SYNOPSIS** `STATUS dummyCallback (void)`

**RETURNS** ERROR.

**SEE ALSO** `winSio`

---

## *dummyCallback()*

**NAME** `dummyCallback()` – dummy callback routine.

**SYNOPSIS** `STATUS dummyCallback (void)`

**RETURNS** ERROR.

**SEE ALSO** `unixSio`

---

## *e()*

**NAME** `e()` – set or display eventpoints (WindView)

**SYNOPSIS** `STATUS e`

```
(
 INSTR * addr, /* where to set eventpoint; 0 means display all */
 event_t eventId, /* event ID */
 int taskNameOrId, /* task affected; 0 means all tasks */
 FUNCPTR evtRtn, /* function to invoke; NULL means no function */
 int arg /* argument to be passed to evtRtn */
)
```

**edi()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | <p>This routine sets "eventpoints"—that is, breakpoint-like instrumentation markers that can be inserted in code to generate and log an event for use with WindView. Event logging must be enabled with <i>wvEvtLogEnable()</i> for the eventpoint to be logged.</p> <p><i>eventId</i> selects the eventpoint number that will be logged: it is in the user event ID range (0-25536).</p> <p>If <i>addr</i> is NULL, then all eventpoints and breakpoints are displayed. If <i>taskNameOrId</i> is 0, then this event is logged in all tasks. The <i>evtRtn</i> routine is called when this eventpoint is hit. If <i>evtRtn</i> returns OK, then the eventpoint is logged; otherwise, it is ignored. If <i>evtRtn</i> is a NULL pointer, then the eventpoint is always logged.</p> <p>Eventpoints are exactly like breakpoints (which are set with the <i>b()</i> command) except in how the system responds when the eventpoint is hit. An eventpoint typically records an event and continues immediately (if <i>evtRtn</i> is supplied, this behavior may be different). Eventpoints cannot be used at interrupt level.</p> <p>To delete an eventpoint, use <i>bd()</i>.</p> |
| <b>RETURNS</b>     | OK, or ERROR if <i>addr</i> is odd or nonexistent in memory, or if the breakpoint table is full.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | <i>dbgLib</i> , <i>wvEvent()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

---

**edi()**

|                    |                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>edi()</i> – return the contents of register <b>edi</b> (also <b>esi</b> – <b>eax</b> ) (i386/i486)                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>int edi (     int taskId /* task ID, 0 means default task */ )</pre>                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This command extracts the contents of register <b>edi</b> from the TCB of a specified task. If <i>taskId</i> is omitted or zero, the last task referenced is assumed.</p> <p>Similar routines are provided for all address registers (<b>edi</b> – <b>eax</b>): <i>edi()</i> – <i>eax()</i>.</p> <p>The stack pointer is accessed via <i>eax()</i>.</p> |
| <b>RETURNS</b>     | The contents of register <b>edi</b> (or the requested register).                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>dbgArchLib</i> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                                        |

---

## *eexattach()*

**NAME** *eexattach()* – publish the *eex* network interface and initialize the driver and device

**SYNOPSIS**

```
STATUS eexattach
(
 int unit, /* unit number */
 int port, /* base I/O address */
 int ivec, /* interrupt vector number */
 int ilevel, /* interrupt level */
 int nTfds, /* # of transmit frames (0=default) */
 int attachment /* 0=default, 1=AUI, 2=BNC, 3=TPE */
)
```

**DESCRIPTION** The routine publishes the *eex* interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state.

**RETURNS** OK or ERROR.

**SEE ALSO** *if\_eex*, *ifLib*

---

## *eexTxStartup()*

**NAME** *eexTxStartup()* – start output on the chip

**SYNOPSIS**

```
#ifdef BSD43_DRIVER static void eexTxStartup
(
 int unit
)
```

**DESCRIPTION** Looks for any action on the queue, and begins output if there is anything there. This routine is called from several possible threads. Each will be described below.

The first, and most common thread, is when a user task requests the transmission of data. Under BSD 4.3, this will cause *eexOutput()* to be called, which will cause *ether\_output()* to be called, which will cause this routine to be called (usually). This routine will not be called if *ether\_output()* finds that our interface output queue is full. In this case, the outgoing data will be thrown out. BSD 4.4 uses a slightly different model in which the generic *ether\_output()* routine is called directly, followed by a call to this routine.

The second, and most obscure thread, is when the reception of certain packets causes an immediate (attempted) response. For example, ICMP echo packets (ping), and ICMP "no listener on that port" notifications. All functions in this driver that handle the reception side are executed in the context of *netTask()*. Always. So, in the case being discussed, *netTask()* will receive these certain packets, cause IP to be stimulated, and cause the generation of a response to be sent. We then find ourselves following the thread explained in the second example, with the important distinction that the context is that of *netTask()*.

The third thread occurs when this routine runs out of TFDs and returns. If this occurs when our output queue is not empty, this routine would typically not get called again until new output was requested. Even worse, if the output queue was also full, this routine would never get called again and we would have a lock state. It DOES happen. To guard against this, the transmit clean-up handler detects the out-of-TFDs state and calls this function. The clean-up handler also runs from *netTask*.

Note that this function is ALWAYS called between an *splnet()* and an *splx()*. This is true because *netTask()*, and *ether\_output()* take care of this when calling this function. Therefore, no calls to these spl functions are needed anywhere in this output thread.

SEE ALSO *if\_eex*

---

## ***eflags()***

**NAME** *eflags()* – return the contents of the status register (i386/i486)

**SYNOPSIS**

```
int eflags
(
 int taskId /* task ID, 0 means default task */
)
```

**DESCRIPTION** This command extracts the contents of the status register from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.

**RETURNS** The contents of the status register.

**SEE ALSO** *dbgArchLib*, *VxWorks Programmer's Guide: Target Shell*

---

## *ei82596EndLoad()*

**NAME** *ei82596EndLoad()* – initialize the driver and device

**SYNOPSIS**

```
END_OBJ *ei82596EndLoad
(
 char * initString /* parameter string */
)
```

**DESCRIPTION** This routine initializes both driver and device to an operational state using the device-specific values specified by *initString*. The *initString* parameter expects an ordered list of colon-separated values.

The format of the *initString* is: *unit:ivec:sysbus:memBase:nTfds:nRfds*

*unit*

Specifies the unit number for this device.

*ivec*

This is the interrupt vector number of the hardware interrupt generated by this Ethernet device. The driver uses *intConnect()* to attach an interrupt handler for this interrupt.

*sysbus*

Passes in values as described in the Intel manual for the 82596. A default number of transmit/receive frames of 32 can be selected by passing zero in the parameters *nTfds* and *nRfds*. In other cases, the number of frames selected should be greater than two.

*memBase*

Informs the driver about the shared memory region. The 82596 shares a region of memory with the driver. The caller of this routine can specify the address of this memory region, or can specify that the driver must obtain this memory region from the system resources. If this parameter is set to the constant "NONE", this routine tries to allocate the shared memory from the system. Any other value for this parameter is interpreted by this routine as the address of the shared memory region to be used.

If the caller provides the shared memory region, the driver assumes that this region does not require cache-coherency operations, nor does it require conversions between virtual and physical addresses. If the caller indicates that this routine must allocate the shared memory region, this routine uses *cacheDmaMalloc()* to obtain some non-cacheable memory. The attributes of this memory are checked, and, if the memory is not both read- and write-coherent, this routine aborts.

**RETURNS** An END object pointer or NULL.

**SEE ALSO** *ei82596End*, *ifLib*, *Intel 82596 User's Manual*

**eiattach()**

---

**eiattach()**

**NAME** *eiattach()* – publish the **ei** network interface and initialize the driver and device

**SYNOPSIS**

```
STATUS eiattach
(
 int unit, /* unit number */
 int ivec, /* interrupt vector number */
 UINT8 sysbus, /* sysbus field of SCP */
 char * memBase, /* address of memory pool or NONE */
 int nTfds, /* no. of transmit frames (0 = default) */
 int nRfds /* no. of receive frames (0 = default) */
)
```

**DESCRIPTION** This routine publishes the **ei** interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state.

The 82596 shares a region of memory with the driver. The caller of this routine can specify the address of this memory region, or can specify that the driver must obtain this memory region from the system resources.

The *sysbus* parameter accepts values as described in the Intel manual for the 82596. A default number of transmit/receive frames of 32 can be selected by passing zero in the parameters *nTfds* and *nRfds*. In other cases, the number of frames selected should be greater than two.

The *memBase* parameter is used to inform the driver about the shared memory region. If this parameter is set to the constant "NONE," then this routine will attempt to allocate the shared memory from the system. Any other value for this parameter is interpreted by this routine as the address of the shared memory region to be used.

If the caller provides the shared memory region, then the driver assumes that this region does not require cache coherency operations, nor does it require conversions between virtual and physical addresses.

If the caller indicates that this routine must allocate the shared memory region, then this routine will use *cacheDmaMalloc()* to obtain some non-cacheable memory. The attributes of this memory will be checked, and if the memory is not both read and write coherent, this routine will abort and return ERROR.

**RETURNS** OK or ERROR.

**SEE ALSO** *if\_ei*, *ifLib*, *Intel 82596 User's Manual*

---

## ***eihkattach()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>eihkattach()</i> – publish the <b>ei</b> network interface and initialize the driver and device                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS eihkattach (     int    unit,      /* unit number */     int    ivec,     /* interrupt vector number */     UINT8  sysbus,   /* sysbus field of SCP */     char * memBase,  /* address of memory pool or NONE */     int    nTfds,    /* no. of transmit frames (0 = default) */     int    nRfds     /* no. of receive frames (0 = default) */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | <p>This routine publishes the <b>ei</b> interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state.</p> <p>The 82596 shares a region of memory with the driver. The caller of this routine can specify the address of this memory region, or can specify that the driver must obtain this memory region from the system resources.</p> <p>The <i>sysbus</i> parameter accepts values as described in the Intel manual for the 82596. A default number of transmit/receive frames of 32 can be selected by passing zero in the parameters <i>nTfds</i> and <i>nRfds</i>. In other cases, the number of frames selected should be greater than two.</p> <p>The <i>memBase</i> parameter is used to inform the driver about the shared memory region. If this parameter is set to the constant "NONE," then this routine will attempt to allocate the shared memory from the system. Any other value for this parameter is interpreted by this routine as the address of the shared memory region to be used.</p> <p>If the caller provides the shared memory region, then the driver assumes that this region does not require cache coherency operations, nor does it require conversions between virtual and physical addresses.</p> <p>If the caller indicates that this routine must allocate the shared memory region, then this routine will use <i>cacheDmaMalloc()</i> to obtain some non-cacheable memory. The attributes of this memory will be checked, and if the memory is not both read and write coherent, this routine will abort and return ERROR.</p> |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <i>if_eihk</i> , <i>ifLib</i> , <i>Intel 82596 User's Manual</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

***eiInt()***

---

***eiInt()***

|                    |                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>eiInt()</i> – entry point for handling interrupts from the 82596                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>void eiInt (     DRV_CTRL * pDrvCtrl )</pre>                                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | The interrupting events are acknowledged to the device, so that the device will deassert its interrupt signal. The amount of work done here is kept to a minimum; the bulk of the work is deferred to the <i>netTask</i> . Several flags are used here to synchronize with task level code and eliminate races. |
| <b>SEE ALSO</b>    | <i>if_eihk</i>                                                                                                                                                                                                                                                                                                  |

---

***eiTxStartup()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>eiTxStartup()</i> – start output on the chip                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>#ifdef BSD43_DRIVER static void eiTxStartup (     int unit )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | <p>Looks for any action on the queue, and begins output if there is anything there. This routine is called from several possible threads. Each will be described below.</p> <p>The first, and most common thread, is when a user task requests the transmission of data. Under BSD 4.3, this will cause <i>eiOutput()</i> to be called, which calls <i>ether_output()</i>, which will usually call this routine. This routine will not be called if <i>ether_output()</i> finds that our interface output queue is full. In this case, the outgoing data will be thrown out. BSD 4.4 uses a slightly different model in which the generic <i>ether_output()</i> routine is called directly, followed by a call to this routine.</p> <p>The second, and most obscure thread, is when the reception of certain packets causes an immediate (attempted) response. For example, ICMP echo packets (ping), and ICMP "no listener on that port" notifications. All functions in this driver that handle the reception side are executed in the context of <i>netTask()</i>. Always. So, in the case being discussed, <i>netTask()</i> will receive these certain packets, cause IP to be stimulated, and cause the generation of a response to be sent. We then find ourselves following the thread</p> |



explained in the second example, with the important distinction that the context is that of *netTask()*.

The third thread occurs when this routine runs out of TFDs and returns. If this occurs when our output queue is not empty, this routine would typically not get called again until new output was requested. Even worse, if the output queue was also full, this routine would never get called again and we would have a lock state. It DOES happen. To guard against this, the transmit clean-up handler detects the out-of-TFDs state and calls this function. The clean-up handler also runs from *netTask*.

Note that this function is ALWAYS called between an *splnet()* and an *splx()*. This is true because *netTask()*, and *ether\_output()* take care of this when calling this function. Therefore, no calls to these spl functions are needed anywhere in this output thread.

SEE ALSO [if\\_eihk](#)

---

## *eiTxStartup()*

**NAME** *eiTxStartup()* – start output on the chip

**SYNOPSIS**

```
void eiTxStartup
(
 DRV_CTRL * pDrvCtrl
)
```

**DESCRIPTION** Looks for any action on the queue, and begins output if there is anything there. This routine is called from several possible threads. Each will be described below.

The first, and most common thread, is when a user task requests the transmission of data. This will cause *eiOutput()* to be called, which will cause *ether\_output()* to be called, which will cause this routine to be called (usually). This routine will not be called if *ether\_output()* finds that our interface output queue is full. In this case, the outgoing data will be thrown out.

The second, and most obscure thread, is when the reception of certain packets causes an immediate (attempted) response. For example, ICMP echo packets (ping), and ICMP "no listener on that port" notifications. All functions in this driver that handle the reception side are executed in the context of *netTask()*. Always. So, in the case being discussed, *netTask()* will receive these certain packets, cause IP to be stimulated, and cause the generation of a response to be sent. We then find ourselves following the thread explained in the second example, with the important distinction that the context is that of *netTask()*.

The third thread occurs when this routine runs out of TFDs and returns. If this occurs when our output queue is not empty, this routine would typically not get called again

until new output was requested. Even worse, if the output queue was also full, this routine would never get called again and we would have a lock state. It DOES happen. To guard against this, the transmit clean-up handler detects the out-of-TFDs state and calls this function. The clean-up handler also runs from netTask.

Note that this function is ALWAYS called between an *splnet()* and an *splx()*. This is true because *netTask()*, and *ether\_output()* take care of this when calling this function. Therefore, no calls to these spl functions are needed anywhere in this output thread.

SEE ALSO *if\_ei*

---

## ***el3c90xEndLoad()***

NAME *el3c90xEndLoad()* – initialize the driver and device

SYNOPSIS

```
END_OBJ * el3c90xEndLoad
(
 char * initString /* String to be parsed by the driver. */
)
```

DESCRIPTION This routine initializes the driver and the device to the operational state. All of the device-specific parameters are passed in *initString*, which expects a string of the following format:

*unit:devMemAddr:devIoAddr:pciMemBase:<vecnum:intLvl:memAdrs:memSize:memWidth:flags:buffMultiplier*

This routine can be called in two modes. If it is called with an empty but allocated string, it places the name of this device (that is, "elPci") into the *initString* and returns 0.

If the string is allocated and not empty, the routine attempts to load the driver using the values specified in the string.

RETURNS An END object pointer, or NULL on error, or 0 and the name of the device if the *initString* was NULL.

SEE ALSO *el3c90xEnd*

---

## *el3c90xInitParse()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>el3c90xInitParse()</i> – parse the initialization string                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre> STATUS <i>el3c90xInitParse</i> (     EL3C90X_DEVICE * <i>pDrvCtrl</i>, /* pointer to the control structure */     char *          <i>initString</i> /* initialization string */ ) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>Parse the input string. This routine is called from <i>el3c90xEndLoad()</i> which initializes some values in the driver control structure with the values passed in the initialization string.</p> <p>The initialization string format is:<br/> <i>unit:devMemAddr:devIoAddr:pciMemBase:&lt;vecNum:intLvl:memAdrs:memSize:memWidth:flags:buffMultiplier</i></p> <p><i>unit</i><br/> Device unit number, a small integer.</p> <p><i>devMemAddr</i><br/> Device register base memory address</p> <p><i>devIoAddr</i><br/> Device register base IO address</p> <p><i>pciMemBase</i><br/> Base address of PCI memory space</p> <p><i>vecNum</i><br/> Interrupt vector number.</p> <p><i>intLvl</i><br/> Interrupt level.</p> <p><i>memAdrs</i><br/> Memory pool address or NONE.</p> <p><i>memSize</i><br/> Memory pool size or zero.</p> <p><i>memWidth</i><br/> Memory system size, 1, 2, or 4 bytes (optional).</p> <p><i>flags</i><br/> Device specific flags, for future use.</p> <p><i>buffMultiplier</i><br/> Buffer Multiplier or NONE. If NONE is specified, it defaults to 2</p> |

**RETURNS** OK, or ERROR if any arguments are invalid.

**SEE ALSO** `el3c90xEnd`

---

## ***elcattach()***

**NAME** `elcattach()` – publish the `elc` network interface and initialize the driver and device

**SYNOPSIS**

```
STATUS elcattach
(
 int unit, /* unit number */
 int ioAddr, /* address of elc's shared memory */
 int ivec, /* interrupt vector to connect to */
 int ilevel, /* interrupt level */
 int memAddr, /* address of elc's shared memory */
 int memSize, /* size of elc's shared memory */
 int config /* 0: RJ45 + AUI(Thick) 1: RJ45 + BNC(Thin) */
)
```

**DESCRIPTION** This routine attaches an `elc` Ethernet interface to the network if the device exists. It makes the interface available by filling in the network interface record. The system will initialize the interface when it is ready to accept packets.

**RETURNS** OK or ERROR.

**SEE ALSO** `if_elc`, `ifLib`, `netShow`

---

## ***elcPut()***

**NAME** `elcPut()` – copy a packet to the interface.

**SYNOPSIS**

```
#ifdef BSD43_DRIVER LOCAL void elcPut
(
 int unit
)
```

**DESCRIPTION** Copy from mbuf chain to transmitter buffer in shared memory.

**SEE ALSO** `if_elc`

---

## *elcShow()*

|                    |                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>elcShow()</i> – display statistics for the SMC 8013WC <b>elc</b> network interface                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>void elcShow (     int unit, /* interface unit */     BOOL zap  /* 1 = zero totals */ )</pre>                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine displays statistics about the <b>elc</b> Ethernet network interface. It has two parameters:</p> <p><i>unit</i><br/>interface unit; should be 0.</p> <p><i>zap</i><br/>if 1, all collected statistics are cleared to zero.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>if_elc</i>                                                                                                                                                                                                                                 |

---

## *elt3c509Load()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>elt3c509Load()</i> – initialize the driver and device                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>END_OBJ * elt3c509Load (     char * initString /* String to be parsed by the driver. */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This routine initializes the driver and the device to the operational state. All of the device-specific parameters are passed in <i>initString</i>, which expects a string of the following format:</p> <p><i>unit:port:intVector:intLevel:attachementType:noRxFrames</i></p> <p>This routine can be called in two modes. If it is called with an empty but allocated string, it places the name of this device (that is, "elt") into the <i>initString</i> and returns 0.</p> <p>If the string is allocated and not empty, the routine attempts to load the driver using the values specified in the string.</p> |

**elt3c509Parse()**

- RETURNS** An END object pointer, or NULL on error, or 0 and the name of the device if the *initString* was NULL.
- SEE ALSO** **elt3c509End**

**elt3c509Parse()**

- NAME** *elt3c509Parse()* – parse the init string
- SYNOPSIS**
- ```
STATUS elt3c509Parse
(
    ELT3C509_DEVICE * pDrvCtrl, /* device pointer */
    char *          initString /* initialization info string */
)
```
- DESCRIPTION** Parse the input string. Fill in values in the driver control structure. The initialization string format is:
- unit:port:intVector:intLevel:attachmentType:noRxFrames*
- unit*
Device unit number, a small integer.
- port*
base I/O address
- intVector*
Interrupt vector number (used with sysIntConnect)
- intLevel*
Interrupt level
- attachmentType*
type of Ethernet connector
- nRxFrames* no. of Rx Frames in integer format
- RETURNS** OK or ERROR for invalid arguments.
- SEE ALSO** **elt3c509End**

eltattach()

NAME *eltattach()* – publish the **elt** interface and initialize the driver and device

SYNOPSIS

```
STATUS eltattach
(
    int    unit,      /* unit number */
    int    port,      /* base I/O address */
    int    ivec,      /* interrupt vector number */
    int    intLevel,  /* interrupt level */
    int    nRxFrames, /* # of receive frames (0=default) */
    int    attachment, /* Ethernet connector to use */
    char * ifName     /* interface name */
)
```

DESCRIPTION The routine publishes the **elt** interface, filling in a network interface record and adding the record to the system list. It also initializes the driver and device to the operational state.

RETURNS OK or ERROR.

SEE ALSO *if_elt*, *ifLib*

eltShow()

NAME *eltShow()* – display statistics for the 3C509 **elt** network interface

SYNOPSIS

```
void eltShow
(
    int unit, /* interface unit */
    BOOL zap  /* 1 = zero totals */
)
```

DESCRIPTION This routine displays statistics about the **elt** Ethernet network interface. It has two parameters:

unit interface unit; should be 0.

zap if 1, all collected statistics are cleared to zero.

RETURNS N/A

SEE ALSO *if_elt*

eltTxOutputStart()

eltTxOutputStart()

| | |
|--------------------|---|
| NAME | <i>eltTxOutputStart()</i> – start output on the board |
| SYNOPSIS | <pre>#ifdef BSD43_DRIVER static void eltTxOutputStart (int unit)</pre> |
| DESCRIPTION | <p>This routine is called from <i>ether_output()</i> when a new packet is enqueued in the interface mbuf queue.</p> <p>Note that this function is ALWAYS called between an <i>splnet()</i> and an <i>splx()</i>. This is true because <i>netTask()</i>, and <i>ether_output()</i> take care of this when calling this function. Therefore, no calls to these spl functions are needed anywhere in this output thread.</p> |
| SEE ALSO | <i>if_elt</i> |

endEtherAddressForm()

| | |
|--------------------|--|
| NAME | <i>endEtherAddressForm()</i> – form an Ethernet address into a packet |
| SYNOPSIS | <pre>M_BLK_ID endEtherAddressForm (M_BLK_ID pMblk, /* pointer to packet mBlk */ M_BLK_ID pSrcAddr, /* pointer to source address */ M_BLK_ID pDstAddr /* pointer to destination address */)</pre> |
| DESCRIPTION | <p>This routine accepts the source and destination addressing information through <i>pSrcAddr</i> and <i>pDstAddr</i> and returns an <i>M_BLK_ID</i> that points to the assembled link-level header. To do this, this routine prefixes the link-level header into the cluster associated with <i>pMblk</i> if there is enough space available in the cluster. It then returns a pointer to the pointer referenced in <i>pMblk</i>. However, if there is not enough space in the cluster associated with <i>pMblk</i>, this call reserves a new mBlk-clBlk-cluster construct for the header information. It then prepends the new mBlk to the mBlk passed in <i>pMblk</i>. As the function value, this routine then returns a pointer to the new mBlk, which is the head of a chain of mBlk structures. The second element in the chain is the mBlk referenced in <i>pMblk</i>.</p> |
| RETURNS | <i>M_BLK_ID</i> or NULL. |
| SEE ALSO | <i>endLib</i> |

endEtherPacketAddrGet()

NAME *endEtherPacketAddrGet()* – locate the addresses in a packet

SYNOPSIS

```
STATUS endEtherPacketAddrGet
(
    M_BLK_ID pMblk, /* pointer to packet */
    M_BLK_ID pSrc,  /* pointer to local source address */
    M_BLK_ID pDst,  /* pointer to local destination address */
    M_BLK_ID pESrc, /* pointer to remote source address (if any) */
    M_BLK_ID pEDst  /* pointer to remote destination address (if any) */
)
```

DESCRIPTION This routine takes a *M_BLK_ID*, locates the address information, and adjusts the *M_BLK_ID* structures referenced in *pSrc*, *pDst*, *pESrc*, and *pEDst* so that their *pData* members point to the addressing information in the packet. The addressing information is not copied. All *mBlk* structures share the same cluster.

RETURNS OK or ERROR.

SEE ALSO *endLib*

endEtherPacketDataGet()

NAME *endEtherPacketDataGet()* – return the beginning of the packet data

SYNOPSIS

```
STATUS endEtherPacketDataGet
(
    M_BLK_ID      pMblk,
    LL_HDR_INFO * pLinkHdrInfo
)
```

DESCRIPTION This routine fills the given *pLinkHdrInfo* with the appropriate offsets.

RETURNS OK or ERROR.

SEE ALSO *endLib*

endFindByName()

| | |
|--------------------|--|
| NAME | <i>endFindByName()</i> – find a device using its string name |
| SYNOPSIS | <pre>END_OBJ* endFindByName (char* pName, /* device name to search for */ int unit)</pre> |
| DESCRIPTION | This routine takes a string name and a unit number and finds the END device that has that name/unit combination. |
| RETURNS | A pointer to an END_OBJ or NULL (if the device is not found). |
| SEE ALSO | muxLib |

endObjFlagSet()

| | |
|--------------------|--|
| NAME | <i>endObjFlagSet()</i> – set the flags member of an END_OBJ structure |
| SYNOPSIS | <pre>STATUS endObjFlagSet (END_OBJ * pEnd, UINT flags)</pre> |
| DESCRIPTION | <p>As input, this routine expects a pointer to an END_OBJ structure (the <i>pEnd</i> parameter) and a flags value (the <i>flags</i> parameter). This routine sets the flags member of the END_OBJ structure to the value of the <i>flags</i> parameter.</p> <p>Because this routine assumes that the driver interface is now up, this routine also sets the attached member of the referenced END_OBJ structure to TRUE.</p> |
| RETURNS | OK |
| SEE ALSO | endLib |

endObjInit()

| | |
|--------------------|---|
| NAME | <i>endObjInit()</i> – initialize an END_OBJ structure |
| SYNOPSIS | <pre>STATUS endObjInit (END_OBJ * pEndObj, /* object to be initialized */ DEV_OBJ* pDevice, /* ptr to device struct */ char * pBaseName, /* device base name, for example, "ln" */ int unit, /* unit number */ NET_FUNCS * pFuncTable, /* END device functions */ char* pDescription) </pre> |
| DESCRIPTION | This routine initializes an END_OBJ structure and fills it with data from the argument list. It also creates and initializes semaphores and protocol list. |
| RETURNS | OK or ERROR. |
| SEE ALSO | endLib |

eneattach()

| | |
|--------------------|---|
| NAME | <i>eneattach()</i> – publish the ene network interface and initialize the driver and device |
| SYNOPSIS | <pre>STATUS eneattach (int unit, /* unit number */ int ioAddr, /* address of ene's shared memory */ int ivec, /* interrupt vector to connect to */ int ilevel /* interrupt level */) </pre> |
| DESCRIPTION | This routine attaches an ene Ethernet interface to the network if the device exists. It makes the interface available by filling in the network interface record. The system will initialize the interface when it is ready to accept packets. |
| RETURNS | OK or ERROR. |
| SEE ALSO | if_ene , ifLib , netShow |

enePut()

enePut()

NAME *enePut()* – copy a packet to the interface.

SYNOPSIS

```
#ifndef BSD43_DRIVER static void enePut
(
    int unit
)
```

DESCRIPTION Copy from mbuf chain to transmitter buffer in shared memory.

SEE ALSO *if_ene*

eneShow()

NAME *eneShow()* – display statistics for the NE2000 **ene** network interface

SYNOPSIS

```
void eneShow
(
    int unit, /* interface unit */
    BOOL zap /* 1 = zero totals */
)
```

DESCRIPTION This routine displays statistics about the **ene** Ethernet network interface. It has two parameters:

unit
interface unit; should be 0.

zap
if 1, all collected statistics are cleared to zero.

RETURNS N/A

SEE ALSO *if_ene*

envLibInit()

| | |
|--------------------|---|
| NAME | <i>envLibInit()</i> – initialize environment variable facility |
| SYNOPSIS | <pre>STATUS envLibInit (BOOL installHooks)</pre> |
| DESCRIPTION | If <i>installHooks</i> is TRUE, task create and delete hooks are installed that will optionally create and destroy private environments for the task being created or destroyed, depending on the state of VX_PRIVATE_ENV in the task options word. If <i>installHooks</i> is FALSE and a task requires a private environment, it is the application's responsibility to create and destroy the private environment, using <i>envPrivateCreate()</i> and <i>envPrivateDestroy()</i> . |
| RETURNS | OK, or ERROR if an environment cannot be allocated or the hooks cannot be installed. |
| SEE ALSO | <i>envLib</i> |

envoy_call_timer()

| | |
|--------------------|--|
| NAME | <i>envoy_call_timer()</i> – execute the specified function when the timer expires |
| SYNOPSIS | <pre>void envoy_call_timer (bits32_t when, void (* what)(void))</pre> |
| DESCRIPTION | This routine executes the <i>what</i> function after <i>when</i> ticks have elapsed. This function is used internally to respond when the interval between the test and set of a "test and set" exceeds the timeout specified by <i>when</i> . |
| RETURNS | N/A |
| SEE ALSO | <i>saIoLib</i> |

envoy_now()

NAME *envoy_now()* – return the number of clock ticks elapsed since the timer was set

SYNOPSIS `bits32_t envoy_now (void)`

DESCRIPTION Call this function to find out the number of clock ticks elapsed since the timer was set.

RETURNS Elapsed time, in ticks.

SEE ALSO `saIoLib`

envPrivateCreate()

NAME *envPrivateCreate()* – create a private environment

SYNOPSIS `STATUS envPrivateCreate`
`(`
`int taskId, /* task to have private environment */`
`int envSource /* -1 = make an empty private environment */`
`/* 0 = copy global to new private env */`
`/* taskId = copy the specified env */`
`)`

DESCRIPTION This routine creates a private set of environment variables for a specified task, if the environment variable task create hook is not installed.

RETURNS OK, or ERROR if memory is insufficient.

SEE ALSO *envLibInit()*, *envPrivateDestroy()*

envPrivateDestroy()

| | |
|--------------------|---|
| NAME | <i>envPrivateDestroy()</i> – destroy a private environment |
| SYNOPSIS | <pre>STATUS envPrivateDestroy (int taskId /* task with private env to destroy */)</pre> |
| DESCRIPTION | This routine destroys a private set of environment variables that were created with <i>envPrivateCreate()</i> . Calling this routine is unnecessary if the environment variable task create hook is installed and the task was spawned with <code>VX_PRIVATE_ENV</code> . |
| RETURNS | OK, or ERROR if the task does not exist. |
| SEE ALSO | <i>envLib</i> , <i>envPrivateCreate()</i> |

envShow()

| | |
|--------------------|--|
| NAME | <i>envShow()</i> – display the environment for a task |
| SYNOPSIS | <pre>void envShow (int taskId /* task for which environment is printed */)</pre> |
| DESCRIPTION | This routine prints to standard output all the environment variables for a specified task. If <i>taskId</i> is NULL, then the calling task's environment is displayed. |
| RETURNS | N/A |
| SEE ALSO | <i>envLib</i> |

errnoGet()

| | |
|--------------------|---|
| NAME | <i>errnoGet()</i> – get the error status value of the calling task |
| SYNOPSIS | <pre>int errnoGet (void)</pre> |
| DESCRIPTION | This routine gets the error status stored in errno . It is provided for compatibility with previous versions of VxWorks and simply accesses errno directly. |
| RETURNS | The error status value contained in errno . |
| SEE ALSO | <i>errnoLib</i> , <i>errnoSet()</i> , <i>errnoOfTaskGet()</i> |

errnoOfTaskGet()

| | |
|--------------------|--|
| NAME | <i>errnoOfTaskGet()</i> – get the error status value of a specified task |
| SYNOPSIS | <pre>int errnoOfTaskGet (int taskId /* task ID, 0 means current task */)</pre> |
| DESCRIPTION | <p>This routine gets the error status most recently set for a specified task. If <i>taskId</i> is zero, the calling task is assumed, and the value currently in errno is returned.</p> <p>This routine is provided primarily for debugging purposes. Normally, tasks access errno directly to set and get their own error status values.</p> |
| RETURNS | The error status of the specified task, or ERROR if the task does not exist. |
| SEE ALSO | <i>errnoLib</i> , <i>errnoSet()</i> , <i>errnoGet()</i> |

errnoOfTaskSet()

| | |
|--------------------|--|
| NAME | <i>errnoOfTaskSet()</i> – set the error status value of a specified task |
| SYNOPSIS | <pre>STATUS errnoOfTaskSet (int taskId, /* task ID, 0 means current task */ int errorValue /* error status value */)</pre> |
| DESCRIPTION | <p>This routine sets the error status for a specified task. If <i>taskId</i> is zero, the calling task is assumed, and errno is set with the specified error status.</p> <p>This routine is provided primarily for debugging purposes. Normally, tasks access errno directly to set and get their own error status values.</p> |
| RETURNS | OK, or ERROR if the task does not exist. |
| SEE ALSO | <i>errnoLib</i> , <i>errnoSet()</i> , <i>errnoOfTaskGet()</i> |

errnoSet()

| | |
|--------------------|---|
| NAME | <i>errnoSet()</i> – set the error status value of the calling task |
| SYNOPSIS | <pre>STATUS errnoSet (int errorValue /* error status value to set */)</pre> |
| DESCRIPTION | <p>This routine sets the errno variable with a specified error status. It is provided for compatibility with previous versions of VxWorks and simply accesses errno directly.</p> |
| RETURNS | OK, or ERROR if the interrupt nest level is too deep. |
| SEE ALSO | <i>errnoLib</i> , <i>errnoGet()</i> , <i>errnoOfTaskSet()</i> |

esmcattach()

NAME *esmcattach()* – publish the **esmc** network interface and initialize the driver.

SYNOPSIS

```
STATUS esmcattach
(
    int unit,      /* unit number */
    int ioAddr,   /* address of esmc's shared memory */
    int intVec,   /* interrupt vector to connect to */
    int intLevel, /* interrupt level */
    int config,   /* 0: Autodetect 1: AUI 2: BNC 3: RJ45 */
    int mode      /* 0: rx in interrupt 1: rx in task(netTask) */
)
```

DESCRIPTION This routine attaches an **esmc** Ethernet interface to the network if the device exists. It makes the interface available by filling in the network interface record. The system will initialize the interface when it is ready to accept packets.

RETURNS OK or ERROR.

SEE ALSO *if_esmc*, *ifLib*, *netShow*

esmcPut()

NAME *esmcPut()* – copy a packet to the interface.

SYNOPSIS

```
#ifdef BSD43_DRIVER LOCAL void esmcPut
(
    int unit
)
```

DESCRIPTION Copy from mbuf chain to transmitter buffer in shared memory.

RETURNS N/A

SEE ALSO *if_esmc*

esmcShow()

| | |
|--------------------|--|
| NAME | <i>esmcShow()</i> – display statistics for the esmc network interface |
| SYNOPSIS | <pre>void esmcShow (int unit, /* interface unit */ BOOL zap /* zero totals */)</pre> |
| DESCRIPTION | <p>This routine displays statistics about the esmc Ethernet network interface. It has two parameters:</p> <p><i>unit</i> interface unit; should be 0.</p> <p><i>zap</i> if 1, all collected statistics are cleared to zero.</p> |
| RETURNS | N/A |
| SEE ALSO | <i>if_esmc</i> |

etherAddrResolve()

| | |
|--------------------|--|
| NAME | <i>etherAddrResolve()</i> – resolve an Ethernet address for a specified Internet address |
| SYNOPSIS | <pre>STATUS etherAddrResolve (struct ifnet * pIf, /* interface on which to send ARP req */ char * targetAddr, /* name or Internet address of target */ char * eHdr, /* where to return the Ethernet addr */ int numTries, /* number of times to try ARPing */ int numTicks /* number of ticks between ARPing */)</pre> |
| DESCRIPTION | <p>This routine uses the Address Resolution Protocol (ARP) and internal ARP cache to resolve the Ethernet address of a machine that owns the Internet address given in <i>targetAddr</i>.</p> |

The first argument *plf* is a pointer to a variable of type **struct ifnet** which identifies the network interface through which the ARP request messages are to be sent out. The routine *ifunit()* is used to retrieve this pointer from the system in the following way:

```
struct ifnet *pIf;  
...  
pIf = ifunit ("ln0");
```

If *ifunit()* returns a non-NULL pointer, it is a valid pointer to the named network interface device structure of type **struct ifnet**. In the above example, *plf* will be pointing to the data structure that describes the first LANCE network interface device if *ifunit()* is successful.

The six-byte Ethernet address is copied to *eHdr*, if the resolution of *targetAddr* is successful. *eHdr* must point to a buffer of at least six bytes.

RETURNS OK if the address is resolved successfully, or ERROR if *eHdr* is NULL, *targetAddr* is invalid, or address resolution is unsuccessful.

SEE ALSO *etherLib*, *etherOutput()*

etherInputHookAdd()

NAME *etherInputHookAdd()* – add a routine to receive all Ethernet input packets

SYNOPSIS

```
STATUS etherInputHookAdd  
(  
    FUNCPTR inputHook, /* routine to receive Ethernet input */  
    char*   pName,     /* name of device if MUX/END is being used */  
    int     unit       /* unit of device if MUX/END is being used */  
)
```

DESCRIPTION This routine adds a hook routine that will be called for every Ethernet packet that is received.

The calling sequence of the input hook routine is:

```
BOOL inputHook  
(  
    struct ifnet *pIf, /* interface packet was received on */  
    char         *buffer, /* received packet */  
    int          length /* length of received packet */  
)
```

The hook routine should return TRUE if it has handled the input packet and no further action should be taken with it. It should return FALSE if it has not handled the input packet and normal processing (for example, Internet) should take place.

The packet is in a temporary buffer when the hook routine is called. This buffer will be reused upon return from the hook. If the hook routine needs to retain the input packet, it should copy it elsewhere.

IMPLEMENTATION A call to the function pointed to by the global function pointer **etherInputHookRtn** should be invoked in the receive routine of every network driver providing this service. For example:

```
...
#include "etherLib.h"
...
xxxRecv ()
...
/* call input hook if any */
if ((etherInputHookRtn != NULL) &&
    (* etherInputHookRtn) (&ls->ls_if, (char *)eh, len))
{
    return; /* input hook has already processed this packet */
}
```

RETURNS OK, always.

SEE ALSO etherLib

etherInputHookDelete()

NAME *etherInputHookDelete()* – delete a network interface input hook routine

SYNOPSIS

```
void etherInputHookDelete
(
    FUNCPTR inputHook,
    char * pName,
    int    unit
)
```

DESCRIPTION This routine deletes a network interface input hook.

RETURNS N/A

SEE ALSO etherLib

etherMultiAdd()

| | |
|--------------------|--|
| NAME | <i>etherMultiAdd()</i> – add multicast address to a multicast address list |
| SYNOPSIS | <pre>int etherMultiAdd (LIST * pList, /* pointer to list of multicast addresses */ char* pAddress /* address you want to add to list */)</pre> |
| DESCRIPTION | This routine adds an Ethernet multicast address list for a given END. The address is a six-byte value pointed to by <i>pAddress</i> . |
| RETURNS | OK or ENETRESET. |
| SEE ALSO | etherMultiLib |

etherMultiDel()

| | |
|--------------------|--|
| NAME | <i>etherMultiDel()</i> – delete an Ethernet multicast address record |
| SYNOPSIS | <pre>int etherMultiDel (LIST * pList, /* pointer to list of multicast addresses */ char* pAddress /* address you want to add to list */)</pre> |
| DESCRIPTION | This routine deletes an Ethernet multicast address from the list. The address is a six-byte value pointed to by <i>pAddress</i> . |
| RETURNS | OK or ENETRESET. |
| SEE ALSO | etherMultiLib |

etherMultiGet()

| | |
|--------------------|--|
| NAME | <i>etherMultiGet()</i> – retrieve a table of multicast addresses from a driver |
| SYNOPSIS | <pre>int etherMultiGet (LIST* pList, /* pointer to list of multicast addresses */ MULTI_TABLE* pTable /* table into which to copy addresses */)</pre> |
| DESCRIPTION | This routine runs down the multicast address list stored in a driver and places all the entries it finds into the multicast table structure passed to it. |
| RETURNS | OK or ERROR. |
| SEE ALSO | <i>etherMultiLib</i> |

etherOutput()

| | |
|--------------------|--|
| NAME | <i>etherOutput()</i> – send a packet on an Ethernet interface |
| SYNOPSIS | <pre>STATUS etherOutput (struct ifnet * pIf, /* interface on which to send */ struct ether_header * pEtherHeader, /* Ethernet header to send */ char * pData, /* data to send */ int dataLength /* # of bytes of data to send */)</pre> |
| DESCRIPTION | <p>This routine sends a packet on the specified Ethernet interface by calling the interface's output routine directly.</p> <p>The first argument <i>pIf</i> is a pointer to a variable of type struct ifnet which contains some useful information about the network interface. A routine named <i>ifunit()</i> can retrieve this pointer from the system in the following way:</p> <pre>struct ifnet *pIf; ... pIf = ifunit ("ln0");</pre> |

If *ifunit()* returns a non-NULL pointer, it is a valid pointer to the named network interface device structure of type **struct ifnet**. In the above example, *plf* points to the data structure that describes the first LANCE network interface device if *ifunit()* is successful.

The second argument *pEtherHeader* should contain a valid Ethernet address of the machine for which the message contained in the argument *pData* is intended. If the Ethernet address of this machine is fixed and well-known to the user, filling in the structure **ether_header** can be accomplished by using *bcopy()* to copy the six-byte Ethernet address into the **ether_dhost** field of the structure **ether_header**. Alternatively, users can make use of the routine *etherAddrResolve()* which will use ARP (Address Resolution Protocol) to resolve the Ethernet address for a specified Internet address.

RETURNS OK, or ERROR if the routine runs out of mbufs.

SEE ALSO *etherLib*, *etherAddrResolve()*

etherOutputHookAdd()

NAME *etherOutputHookAdd()* – add a routine to receive all Ethernet output packets

SYNOPSIS

```
STATUS etherOutputHookAdd
(
    FUNCPTR outputHook /* routine to receive Ethernet output */
)
```

DESCRIPTION This routine adds a hook routine that will be called for every Ethernet packet that is transmitted.

The calling sequence of the output hook routine is:

```
BOOL outputHook
(
    struct ifnet *pIf, /* interface packet will be sent on */
    char *buffer, /* packet to transmit */
    int length /* length of packet to transmit */
)
```

The hook is called immediately before transmission. The hook routine should return TRUE if it has handled the output packet and no further action should be taken with it. It should return FALSE if it has not handled the output packet and normal transmission should take place.

The Ethernet packet data is in a temporary buffer when the hook routine is called. This buffer will be reused upon return from the hook. If the hook routine needs to retain the output packet, it should be copied elsewhere.

IMPLEMENTATION A call to the function pointed to be the global function pointer **etherOutputHookRtn** should be invoked in the transmit routine of every network driver providing this service. For example:

```
...
#include "etherLib.h"
...
xxxStartOutput ()
/* call output hook if any */
if ((etherOutputHookRtn != NULL) &&
    (* etherOutputHookRtn) (&ls->ls_if, buf0, len))
{
    /* output hook has already processed this packet */
}
else
...

```

RETURNS OK, if the hook could be added, ERROR otherwise.

SEE ALSO etherLib

etherOutputHookDelete()

NAME *etherOutputHookDelete()* – delete a network interface output hook routine

SYNOPSIS

```
void etherOutputHookDelete
(
    FUNCPTR outputHook
)
```

DESCRIPTION This routine deletes a network interface output hook, which must be supplied as the only argument.

RETURNS N/A

SEE ALSO etherLib

etherTypeGet()

- NAME** *etherTypeGet()* – get the type from an ethernet packet
- SYNOPSIS**
- ```
USHORT etherTypeGet
(
 char * pPacket /* pointer to the beginning of the packet */
)
```
- DESCRIPTION** This routine returns a short that is the ethertype (defined in RFC 1700) from either an 802.3 addressed packet or an RFC 894 packet. Most packets are encoded as described in RFC 894 but we should also be able to understand 802.3 addressing.
- RETURNS** A USHORT value that is the ethertype, or 0 on error.
- SEE ALSO** *etherLib*, *RFC 894*, *TCP/IP Illustrated*, Volume 1, by Richard Stevens.
- 

## ***evbNs16550HrdInit()***

- NAME** *evbNs16550HrdInit()* – initialize the NS 16550 chip
- SYNOPSIS**
- ```
void evbNs16550HrdInit  
(  
    EVBNS16550_CHAN * pChan  
)
```
- DESCRIPTION** This routine is called to reset the NS 16550 chip to a quiescent state.
- SEE ALSO** *evbNs16550Sio*

evbNs16550Int()

- NAME** *evbNs16550Int()* – handle a receiver/transmitter interrupt for the NS 16550 chip
- SYNOPSIS**
- ```
void evbNs16550Int
(
 EVBNS16550_CHAN * pChan
)
```
- DESCRIPTION** This routine is called to handle interrupts. If there is another character to be transmitted, it sends it. If the interrupt handler is called erroneously (for example, if a device has never been created for the channel), it disables the interrupt.
- SEE ALSO** *evbNs16550Sio*

---

## *excConnect()*

- NAME** *excConnect()* – connect a C routine to an exception vector (PowerPC)
- SYNOPSIS**
- ```
STATUS excConnect
(
    VOIDFUNCPTR * vector, /* exception vector to attach to */
    VOIDFUNCPTR routine /* routine to be called */
)
```
- DESCRIPTION** This routine connects a specified C routine to a specified exception vector. An exception stub is created and in placed at *vector* in the exception table. The address of *routine* is stored in the exception stub code. When an exception occurs, the processor jumps to the exception stub code, saves the registers, and calls the C routines.
- The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.
- The registers are saved to an Exception Stack Frame (ESF) placed on the stack of the task that has produced the exception. The structure of the ESF used to save the registers is defined in **h/arch/ppc/esfPpc.h**.
- The only argument passed by the exception stub to the C routine is a pointer to the ESF containing the registers values. The prototype of this C routine is described below:

```
void excHandler (ESFPPC *);
```

When the C routine returns, the exception stub restores the registers saved in the ESF and continues execution of the current task.

RETURNS OK, always.

SEE ALSO *excArchLib*, *excIntConnect()*, *excVecSet()*

excCrtConnect()

NAME *excCrtConnect()* – connect a C routine to a critical exception vector (PowerPC 403)

SYNOPSIS

```
STATUS excCrtConnect
(
    VOIDFUNCPTR * vector, /* exception vector to attach to */
    VOIDFUNCPTR routine /* routine to be called */
)
```

DESCRIPTION This routine connects a specified C routine to a specified critical exception vector. An exception stub is created and in placed at *vector* in the exception table. The address of *routine* is stored in the exception stub code. When an exception occurs, the processor jumps to the exception stub code, saves the registers, and call the C routines.

The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.

The registers are saved to an Exception Stack Frame (ESF) which is placed on the stack of the task that has produced the exception. The ESF structure is defined in ***h/arch/ppc/esfPpc.h***.

The only argument passed by the exception stub to the C routine is a pointer to the ESF containing the register values. The prototype of this C routine is as follows:

```
void excHandler (ESFPPC *);
```

When the C routine returns, the exception stub restores the registers saved in the ESF and continues execution of the current task.

RETURNS OK, always.

SEE ALSO *excArchLib*, *excIntConnect()*, *excIntCrtConnect*, *excVecSet()*

excHookAdd()

NAME *excHookAdd()* – specify a routine to be called with exceptions

SYNOPSIS

```
void excHookAdd
(
    FUNCPTR excepHook /* routine to call when exceptions occur */
)
```

DESCRIPTION This routine specifies a routine that will be called when hardware exceptions occur. The specified routine is called after normal exception handling, which includes displaying information about the error. Upon return from the specified routine, the task that incurred the error is suspended.

The exception handling routine should be declared as:

```
void myHandler
(
    int      task,      /* ID of offending task           */
    int      vecNum,   /* exception vector number       */
    ESFxx *pEsf      /* pointer to exception stack frame */
)
```

where *task* is the ID of the task that was running when the exception occurred. *ESFxx* is architecture-specific and can be found by examining */target/h/arch/arch/esfarch.h*; for example, the PowerPC uses *ESFPPC*.

This facility is normally used by **dbgLib()** to activate its exception handling mechanism. If an application provides its own exception handler, it will supersede the **dbgLib** mechanism.

RETURNS N/A

SEE ALSO *excLib*, *excTask()*

excInit()

NAME *excInit()* – initialize the exception handling package

SYNOPSIS `STATUS excInit ()`

excIntConnect()

| | |
|--------------------|---|
| DESCRIPTION | This routine installs the exception handling facilities and spawns <i>excTask()</i> , which performs special exception handling functions that need to be done at task level. It also creates the message queue used to communicate with <i>excTask()</i> . |
| NOTE | The exception handling facilities should be installed as early as possible during system initialization in the root task, <i>usrRoot()</i> , in <i>usrConfig.c</i> . |
| RETURNS | OK, or ERROR if a message queue cannot be created or <i>excTask()</i> cannot be spawned. |
| SEE ALSO | <i>excLib</i> , <i>excTask()</i> |

excIntConnect()

| | |
|--------------------|--|
| NAME | <i>excIntConnect()</i> – connect a C routine to an asynchronous exception vector (PowerPC, ARM) |
| SYNOPSIS | <pre> STATUS excIntConnect (VOIDFUNCPTR * vector, /* exception vector to attach to */ VOIDFUNCPTR routine /* routine to be called */) </pre> |
| DESCRIPTION | <p>This routine connects a specified C routine to a specified asynchronous exception vector. When the C routine is invoked, interrupts are still locked. It is the responsibility of the C routine to re-enable the interrupt.</p> <p>The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.</p> |
| NOTE | <p>On PowerPC, the vector is typically the external interrupt vector 0x500 and the decremter vector 0x900. An interrupt stub is created and placed at <i>vector</i> in the exception table. The address of <i>routine</i> is stored in the interrupt stub code. When the asynchronous exception occurs the processor jumps to the interrupt stub code, saves only the requested registers, and calls the C routines.</p> <p>Before saving the requested registers, the interrupt stub switches from the current task stack to the interrupt stack. For nested interrupts, no stack-switching is performed, because the interrupt is already set.</p> |
| NOTE | <p>On the ARM, the address of <i>routine</i> is stored in a function pointer to be called by the stub installed on the IRQ exception vector following an asynchronous exception. This routine is responsible for determining the interrupt source and despatching the correct handler for that source.</p> |

Before calling the routine, the interrupt stub switches to SVC mode, changes to a separate interrupt stack and saves necessary registers. In the case of a nested interrupt, no SVC stack switch occurs.

RETURNS OK, always.

SEE ALSO `excArchLib`, `excConnect()`, `excVecSet()`

excIntCrtConnect()

NAME `excIntCrtConnect()` – connect a C routine to a critical interrupt vector (PowerPC 403)

SYNOPSIS

```
STATUS excIntCrtConnect
(
    VOIDFUNCPTR * vector, /* exception vector to attach to */
    VOIDFUNCPTR routine /* routine to be called */
)
```

DESCRIPTION This routine connects a specified C routine to a specified asynchronous critical exception vector such as the critical external interrupt vector (0x100), or the watchdog timer vector (0x1020). An interrupt stub is created and placed at *vector* in the exception table. The address of *routine* is stored in the interrupt stub code. When the asynchronous exception occurs, the processor jumps to the interrupt stub code, saves only the requested registers, and calls the C routines.

When the C routine is invoked, interrupts are still locked. It is the C routine's responsibility to re-enable interrupts.

The routine can be any normal C routine, except that it must not invoke certain operating system functions that may block or perform I/O operations.

Before the requested registers are saved, the interrupt stub switches from the current task stack to the interrupt stack. In the case of nested interrupts, no stack switching is performed, because the interrupt stack is already set.

RETURNS OK, always.

SEE ALSO `excArchLib`, `excConnect()`, `excCrtConnect`, `excVecSet()`

excTask()

| | |
|--------------------|--|
| NAME | <i>excTask()</i> – handle task-level exceptions |
| SYNOPSIS | <code>void excTask ()</code> |
| DESCRIPTION | This routine is spawned as a task by <i>excInit()</i> to perform functions that cannot be performed at interrupt or trap level. It has a priority of 0. Do not suspend, delete, or change the priority of this task. |
| RETURNS | N/A |
| SEE ALSO | <i>excLib</i> , <i>excInit()</i> |

excVecGet()

| | |
|--------------------|--|
| NAME | <i>excVecGet()</i> – get a CPU exception vector (PowerPC, ARM) |
| SYNOPSIS | <pre>FUNCPTR excVecGet (FUNCPTR * vector /* vector offset */)</pre> |
| DESCRIPTION | This routine returns the address of the C routine currently connected to <i>vector</i> . |
| RETURNS | The address of the C routine. |
| SEE ALSO | <i>excArchLib</i> , <i>excVecSet()</i> |

excVecInit()

| | |
|-----------------|--|
| NAME | <i>excVecInit()</i> – initialize the exception/interrupt vectors |
| SYNOPSIS | <code>STATUS excVecInit (void)</code> |

| | |
|--------------------|--|
| DESCRIPTION | <p>This routine sets all exception vectors to point to the appropriate default exception handlers. These handlers will safely trap and report exceptions caused by program errors or unexpected hardware interrupts.</p> <p>MC680x0: All vectors from vector 2 (address 0x0008) to 255 (address 0x03fc) are initialized. Vectors 0 and 1 contain the reset stack pointer and program counter.</p> <p>SPARC All vectors from 0 (offset 0x000) through 255 (offset 0xff0) are initialized.</p> <p>i960: The i960 fault table is filled with a default fault handler, and all non-reserved vectors in the i960 interrupt table are filled with a default interrupt handler.</p> <p>MIPS All MIPS exception, trap, and interrupt vectors are set to default handlers.</p> <p>i386/i486: All vectors from vector 0 (address 0x0000) to 255 (address 0x07f8) are initialized to default handlers.</p> <p>PowerPC: There are 48 vectors and only vectors that are used are initialized.</p> <p>ARM All exception vectors are initialized to default handlers except 0x14 (Address) which is now reserved on the ARM and 0x1C (FIQ), which is not used by VxWorks.</p> |
| NOTE | <p>This routine is usually called from the system start-up routine, <i>usrInit()</i>, in <i>usrConfig.c</i>. It must be called before interrupts are enabled. (SPARC: It must also be called when the system runs with the on-chip windows (no stack)).</p> |
| RETURNS | OK, always. |
| SEE ALSO | <i>excArchLib</i> , <i>excLib</i> |

excVecSet()

| | |
|-----------------|--|
| NAME | <i>excVecSet()</i> – set a CPU exception vector (PowerPC, ARM) |
| SYNOPSIS | <pre>void excVecSet (FUNCPTR * vector, /* vector offset */ FUNCPTR function /* address to place in vector */)</pre> |

exit()

| | |
|--------------------|--|
| DESCRIPTION | This routine specifies the C routine that will be called when the exception corresponding to <i>vector</i> occurs. This routine does not create the exception stub; it simply replaces the C routine to be called in the exception stub. |
| NOTE ARM | On the ARM, there is no <i>excConnect()</i> routine, unlike the PowerPC. The C routine is attached to a default stub using <i>excVecSet()</i> . |
| RETURNS | N/A |
| SEE ALSO | <i>excArchLib</i> , <i>excVecGet()</i> , <i>excConnect()</i> , <i>excIntConnect()</i> |

exit()

| | |
|--------------------|---|
| NAME | <i>exit()</i> – exit a task (ANSI) |
| SYNOPSIS | <pre>void exit (int code /* code stored in TCB for delete hooks */)</pre> |
| DESCRIPTION | This routine is called by a task to cease to exist as a task. It is called implicitly when the "main" routine of a spawned task is exited. The <i>code</i> parameter will be stored in the WIND_TCB for possible use by the delete hooks, or post-mortem debugging. |
| ERRNO | N/A |
| SEE ALSO | <i>taskLib</i> , <i>taskDelete()</i> , <i>American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: Input/Output (stdlib.h)</i> , <i>VxWorks Programmer's Guide: Basic OS</i> |

exp()

| | |
|-----------------|---|
| NAME | <i>exp()</i> – compute an exponential value (ANSI) |
| SYNOPSIS | <pre>double exp (double x /* exponent */)</pre> |

| | |
|----------------------|---|
| DESCRIPTION | This routine returns the exponential value of x in double precision (IEEE double, 53 bits). A range error occurs if x is too large. |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision exponential value of x . Special cases: If x is +INF or NaN, <i>exp()</i> returns x . If x is -INF, it returns 0. |
| SEE ALSO | ansiMath, mathALib |

expf()

| | |
|----------------------|--|
| NAME | <i>expf()</i> – compute an exponential value (ANSI) |
| SYNOPSIS | <pre>float expf (float x /* exponent */)</pre> |
| DESCRIPTION | This routine returns the exponential of x in single precision. |
| INCLUDE FILES | math.h |
| RETURNS | The single-precision exponential value of x . |
| SEE ALSO | mathALib |

fabs()

| | |
|-----------------|--|
| NAME | <i>fabs()</i> – compute an absolute value (ANSI) |
| SYNOPSIS | <pre>double fabs (double v /* number to return the absolute value of */)</pre> |

| | |
|----------------------|--|
| DESCRIPTION | This routine returns the absolute value of <i>v</i> in double precision. |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision absolute value of <i>v</i> . |
| ERRNO | EDOM, ERANGE |
| SEE ALSO | ansiMath, mathALib |

fabsf()

| | |
|----------------------|---|
| NAME | <i>fabsf()</i> – compute an absolute value (ANSI) |
| SYNOPSIS | <pre>float fabsf (float v /* number to return the absolute value of */)</pre> |
| DESCRIPTION | This routine returns the absolute value of <i>v</i> in single precision. |
| INCLUDE FILES | math.h |
| RETURNS | The single-precision absolute value of <i>v</i> . |
| SEE ALSO | mathALib |

fclose()

| | |
|--------------------|--|
| NAME | <i>fclose()</i> – close a stream (ANSI) |
| SYNOPSIS | <pre>int fclose (FILE * fp /* stream to close */)</pre> |
| DESCRIPTION | This routine flushes a specified stream and closes the associated file. Any unwritten buffered data is delivered to the host environment to be written to the file; any unread |

buffered data is discarded. The stream is disassociated from the file. If the associated buffer was allocated automatically, it is deallocated.

| | |
|---------------|--|
| INCLUDE FILES | <code>stdio.h</code> |
| RETURNS | Zero if the stream is closed successfully, or EOF if errors occur. |
| ERRNO | EBADF |
| SEE ALSO | <code>ansiStdio</code> , <code>fflush()</code> |

fdDevCreate()

NAME *fdDevCreate()* – create a device for a floppy disk

SYNOPSIS

```
BLK_DEV *fdDevCreate
(
    int drive,      /* driver number of floppy disk (0 - 3) */
    int fdType,     /* type of floppy disk */
    int nBlocks,   /* device size in blocks (0 = whole disk) */
    int blkOffset /* offset from start of device */
)
```

DESCRIPTION This routine creates a device for a specified floppy disk.

The *drive* parameter is the drive number of the floppy disk; valid values are 0 to 3.

The *fdType* parameter specifies the type of diskette, which is described in the structure table **fdTypes[]** in `sysLib.c`. *fdType* is an index to the table. Currently the table contains two diskette types:

- An *fdType* of 0 indicates the first entry in the table (3.5" 2HD, 1.44MB);
- An *fdType* of 1 indicates the second entry in the table (5.25" 2HD, 1.2MB).

Members of the **fdTypes[]** structure are:

```
int  sectors;      /* no of sectors */
int  sectorsTrack; /* sectors per track */
int  heads;        /* no of heads */
int  cylinders;    /* no of cylinders */
int  secSize;      /* bytes per sector, 128 << secSize */
char gap1;         /* gap1 size for read, write */
char gap2;         /* gap2 size for format */
char dataRate;     /* data transfer rate */
```

fdDrv()

```

char stepRate;      /* stepping rate */
char headUnload;   /* head unload time */
char headLoad;     /* head load time */
char mfm;          /* MFM bit for read, write, format */
char sk;           /* SK bit for read */
char *name;        /* name */

```

The *nBlocks* parameter specifies the size of the device, in blocks. If *nBlocks* is zero, the whole disk is used.

The *blkOffset* parameter specifies an offset, in blocks, from the start of the device to be used when writing or reading the floppy disk. This offset is added to the block numbers passed by the file system during disk accesses. (VxWorks file systems always use block numbers beginning at zero for the start of a device.) Normally, *blkOffset* is 0.

RETURNS A pointer to a block device structure (BLK_DEV) or NULL if memory cannot be allocated for the device structure.

SEE ALSO *nec765Fd*, *fdDrv()*, *fdRawio()*, *dosFsMkfs()*, *dosFsDevInit()*, *rt11FsDevInit()*, *rt11FsMkfs()*, *rawFsDevInit()*

fdDrv()

NAME *fdDrv()* – initialize the floppy disk driver

SYNOPSIS

```

STATUS fdDrv
(
    int vector, /* interrupt vector */
    int level   /* interrupt level */
)

```

DESCRIPTION This routine initializes the floppy driver, sets up interrupt vectors, and performs hardware initialization of the floppy chip.

This routine should be called exactly once, before any reads, writes, or calls to *fdDevCreate()*. Normally, it is called by *usrRoot()* in *usrConfig.c*.

RETURNS OK.

SEE ALSO *nec765Fd*, *fdDevCreate()*, *fdRawio()*

fdopen()

| | |
|----------------------|--|
| NAME | <i>fdopen()</i> – open a file specified by a file descriptor (POSIX) |
| SYNOPSIS | <pre>FILE * fdopen (int fd, /* file descriptor */ const char * mode /* mode to open with */)</pre> |
| DESCRIPTION | This routine opens the file specified by the file descriptor <i>fd</i> and associates a stream with it. The <i>mode</i> argument is used just as in the <i>fopen()</i> function. |
| INCLUDE FILES | <code>stdio.h</code> |
| RETURNS | A pointer to a stream, or a null pointer if an error occurs, with errno set to indicate the error. |
| ERRNO | EINVAL |
| SEE ALSO | <i>ansiStdio</i> , <i>fopen()</i> , <i>freopen()</i> , <i>.br Information Technology – POSIX – Part 1: System API [C Language]</i> , <i>IEEE Std 1003.1</i> |

fdprintf()

| | |
|--------------------|---|
| NAME | <i>fdprintf()</i> – write a formatted string to a file descriptor |
| SYNOPSIS | <pre>int fdprintf (int fd, /* file descriptor to write to */ const char * fmt /* format string to write */)</pre> |
| DESCRIPTION | This routine writes a formatted string to a specified file descriptor. Its function and syntax are otherwise identical to <i>printf()</i> . |
| RETURNS | The number of characters output, or ERROR if there is an error during output. |
| SEE ALSO | <i>fiolib</i> , <i>printf()</i> |

fdRawio()

fdRawio()**NAME** *fdRawio()* – provide raw I/O access

SYNOPSIS

```
STATUS fdRawio
(
    int      drive, /* drive number of floppy disk (0 - 3) */
    int      fdType, /* type of floppy disk */
    FD_RAW * pFdRaw /* pointer to FD_RAW structure */
)
```

DESCRIPTION This routine is called when the raw I/O access is necessary.

The *drive* parameter is the drive number of the floppy disk; valid values are 0 to 3.

The *fdType* parameter specifies the type of diskette, which is described in the structure table **fdTypes[]** in **sysLib.c**. *fdType* is an index to the table. Currently the table contains two diskette types:

- An *fdType* of 0 indicates the first entry in the table (3.5" 2HD, 1.44MB);
- An *fdType* of 1 indicates the second entry in the table (5.25" 2HD, 1.2MB).

The *pFdRaw* is a pointer to the structure **FD_RAW**, defined in **nec765Fd.h**

RETURNS OK or ERROR.**SEE ALSO** **nec765Fd**, **fdDrv()**, **fdDevCreate()**

fei82557EndLoad()**NAME** *fei82557EndLoad()* – initialize the driver and device

SYNOPSIS

```
END_OBJ* fei82557EndLoad
(
    char * initString /* parameter string */
)
```

DESCRIPTION This routine initializes both, driver and device to an operational state using device specific parameters specified by *initString*.

The parameter string, *initString*, is an ordered list of parameters each separated by a colon. The format of *initString* is, "*unit:memBase:memSize:nCFDs:nRFDs:flags*"

The 82557 shares a region of memory with the driver. The caller of this routine can specify the address of this memory region, or can specify that the driver must obtain this memory region from the system resources.

A default number of transmit/receive frames of 32 can be selected by passing zero in the parameters *nTfds* and *nRfds*. In other cases, the number of frames selected should be greater than two.

The *memBase* parameter is used to inform the driver about the shared memory region. If this parameter is set to the constant "NONE," then this routine will attempt to allocate the shared memory from the system. Any other value for this parameter is interpreted by this routine as the address of the shared memory region to be used. The *memSize* parameter is used to check that this region is large enough with respect to the provided values of both transmit/receive frames.

If the caller provides the shared memory region, then the driver assumes that this region does not require cache coherency operations, nor does it require conversions between virtual and physical addresses.

If the caller indicates that this routine must allocate the shared memory region, then this routine will use *cacheDmaMalloc()* to obtain some non-cacheable memory. The attributes of this memory will be checked, and if the memory is not write coherent, this routine will abort and return ERROR.

RETURNS an END object pointer, or NULL on error.

SEE ALSO *fei82557End*, *ifLib*, *Intel 82557 User's Manual*

feiattach()

NAME *feiattach()* – publish the fei network interface

SYNOPSIS

```
STATUS feiattach
(
    int    unit,      /* unit number */
    char * memBase, /* address of shared memory (NONE = malloc) */
    int    nCFD,     /* command frames (0 = default) */
    int    nRFD,     /* receive frames (0 = default) */
    int    nRFDLoan /* loanable rx frames (0 = default, -1 = 0) */
)
```

DESCRIPTION This routine publishes the fei interface by filling in a network interface record and adding the record to the system list.

feof()

The 82557 shares a region of main memory with the CPU. The caller of this routine can specify the address of this shared memory region through the *memBase* parameter; if *memBase* is set to the constant **NONE**, the driver will allocate the shared memory region.

If the caller provides the shared memory region, the driver assumes that this region does not require cache coherency operations.

If the caller indicates that *feiattach()* must allocate the shared memory region, *feiattach()* will use *cacheDmaMalloc()* to obtain a block of non-cacheable memory. The attributes of this memory will be checked, and if the memory is not both read and write coherent, *feiattach()* will abort and return **ERROR**.

A default number of 32 command (transmit) and 32 receive frames can be selected by passing zero in the parameters *nCFD* and *nRFD*, respectively. If *nCFD* or *nRFD* is used to select the number of frames, the values should be greater than two.

A default number of 8 loanable receive frames can be selected by passing zero in the parameters *nRFDLoan*, else set *nRFDLoan* to the desired number of loanable receive frames. If *nRFDLoan* is set to -1, no loanable receive frames will be allocated/used.

RETURNS OK, or **ERROR** if the driver could not be published and initialized.

SEE ALSO *if_fei*, *ifLib*, *Intel 82557 User's Manual*

feof()

NAME *feof()* – test the end-of-file indicator for a stream (ANSI)

SYNOPSIS

```
int feof
(
    FILE * fp /* stream to test */
)
```

DESCRIPTION This routine tests the end-of-file indicator for a specified stream.

INCLUDE FILES *stdio.h*

RETURNS Non-zero if the end-of-file indicator is set for *fp*.

SEE ALSO *ansiStdio*, *clearerr()*

ferror()

| | |
|----------------------|---|
| NAME | <i>ferror()</i> – test the error indicator for a file pointer (ANSI) |
| SYNOPSIS | <pre>int ferror (FILE * fp /* stream to test */)</pre> |
| DESCRIPTION | This routine tests the error indicator for the stream pointed to by <i>fp</i> . |
| INCLUDE FILES | <code>stdio.h</code> |
| RETURNS | Non-zero if the error indicator is set for <i>fp</i> . |
| SEE ALSO | <code>ansiStdio</code> , <code>clearerr()</code> |

fflush()

| | |
|----------------------|---|
| NAME | <i>fflush()</i> – flush a stream (ANSI) |
| SYNOPSIS | <pre>int fflush (FILE * fp /* stream to flush */)</pre> |
| DESCRIPTION | This routine writes to the file any unwritten data for a specified output or update stream for which the most recent operation was not input; for an input stream the behavior is undefined. |
| CAVEAT | ANSI specifies that if <i>fp</i> is a null pointer, <i>fflush()</i> performs the flushing action on all streams for which the behavior is defined; however, this is not implemented in VxWorks. |
| INCLUDE FILES | <code>stdio.h</code> |
| RETURNS | Zero, or EOF if a write error occurs. |
| ERRNO | EBADF |
| SEE ALSO | <code>ansiStdio</code> , <code>fclose()</code> |

fgetc()

| | |
|----------------------|--|
| NAME | <i>fgetc()</i> – return the next character from a stream (ANSI) |
| SYNOPSIS | <pre>int fgetc (FILE * fp /* stream to read from */)</pre> |
| DESCRIPTION | <p>This routine returns the next character (converted to an int) from the specified stream, and advances the file position indicator for the stream.</p> <p>If the stream is at end-of-file, the end-of-file indicator for the stream is set; if a read error occurs, the error indicator is set.</p> |
| INCLUDE FILES | stdio.h |
| RETURNS | The next character from the stream, or EOF if the stream is at end-of-file or a read error occurs. |
| SEE ALSO | ansiStdio, fgets(), getc() |

fgetpos()

| | |
|----------------------|---|
| NAME | <i>fgetpos()</i> – store the current value of the file position indicator for a stream (ANSI) |
| SYNOPSIS | <pre>int fgetpos (FILE * fp, /* stream */ fpos_t * pos /* where to store position */)</pre> |
| DESCRIPTION | <p>This routine stores the current value of the file position indicator for a specified stream <i>fp</i> in the object pointed to by <i>pos</i>. The value stored contains unspecified information usable by <i>fsetpos()</i> for repositioning the stream to its position at the time <i>fgetpos()</i> was called.</p> |
| INCLUDE FILES | stdio.h |
| RETURNS | Zero, or non-zero if unsuccessful, with errno set to indicate the error. |
| SEE ALSO | ansiStdio, fsetpos() |

fgets()

| | |
|----------------------|---|
| NAME | <i>fgets()</i> – read a specified number of characters from a stream (ANSI) |
| SYNOPSIS | <pre>char * fgets (char * buf, /* where to store characters */ size_t n, /* no. of bytes to read + 1 */ FILE * fp /* stream to read from */)</pre> |
| DESCRIPTION | <p>This routine stores in the array <i>buf</i> up to <i>n</i>-1 characters from a specified stream. No additional characters are read after a new-line or end-of-line. A null character is written immediately after the last character read into the array.</p> <p>If end-of-file is encountered and no characters have been read, the contents of the array remain unchanged. If a read error occurs, the array contents are indeterminate.</p> |
| INCLUDE FILES | <code>stdio.h</code> |
| RETURNS | A pointer to <i>buf</i> , or a null pointer if an error occurs or end-of-file is encountered and no characters have been read. |
| SEE ALSO | <code>ansiStdio</code> , <code>fread()</code> , <code>fgetc()</code> |

fileno()

| | |
|----------------------|---|
| NAME | <i>fileno()</i> – return the file descriptor for a stream (POSIX) |
| SYNOPSIS | <pre>int fileno (FILE * fp /* stream */)</pre> |
| DESCRIPTION | This routine returns the file descriptor associated with a specified stream. |
| INCLUDE FILES | <code>stdio.h</code> |
| RETURNS | The file descriptor, or -1 if an error occurs, with <code>errno</code> set to indicate the error. |
| SEE ALSO | <code>ansiStdio</code> , <i>Information Technology – POSIX – Part 1: System API [C Lang.]</i> , IEEE Std 1003.1 |

fioFormatV()

NAME *fioFormatV()* – convert a format string

SYNOPSIS

```
int fioFormatV
(
    const char * fmt,          /* format string */
    va_list    vaList,       /* pointer to varargs list */
    FUNCPTR    outRoutine,   /* handler for args as they're formatted */
    int        outarg        /* argument to routine */
)
```

DESCRIPTION This routine is used by the *printf()* family of routines to handle the actual conversion of a format string. The first argument is a format string, as described in the entry for *printf()*. The second argument is a variable argument list *vaList* that was previously established.

As the format string is processed, the result will be passed to the output routine whose address is passed as the third parameter, *outRoutine*. This output routine may output the result to a device, or put it in a buffer. In addition to the buffer and length to output, the fourth argument, *outarg*, will be passed through as the third parameter to the output routine. This parameter could be a file descriptor, a buffer address, or any other value that can be passed in an "int".

The output routine should be declared as follows:

```
STATUS outRoutine
(
    char *buffer, /* buffer passed to routine */
    int  nchars, /* length of buffer */
    int  outarg  /* arbitrary arg passed to fmt routine */
)
```

The output routine should return OK if successful, or ERROR if unsuccessful.

RETURNS The number of characters output, or ERROR if the output routine returned ERROR.

SEE ALSO *fioLib*

fiOLibInit()

| | |
|--------------------|--|
| NAME | <i>fiOLibInit()</i> – initialize the formatted I/O support library |
| SYNOPSIS | <code>void fiOLibInit (void)</code> |
| DESCRIPTION | This routine initializes the formatted I/O support library. It should be called once in <i>usrRoot()</i> when formatted I/O functions such as <i>printf()</i> and <i>scanf()</i> are used. |
| RETURNS | N/A |
| SEE ALSO | <i>fiOLib</i> |

fiORdString()

| | |
|--------------------|--|
| NAME | <i>fiORdString()</i> – read a string from a file |
| SYNOPSIS | <pre>int fiORdString (int fd, /* fd of device to read */ char string[], /* buffer to receive input */ int maxbytes /* max no. of chars to read */)</pre> |
| DESCRIPTION | This routine puts a line of input into <i>string</i> . The specified input file descriptor is read until <i>maxbytes</i> , an EOF, an EOS, or a newline character is reached. A newline character or EOF is replaced with EOS, unless <i>maxbytes</i> characters have been read. |
| RETURNS | The length of the string read, including the terminating EOS; or EOF if a read error occurred or end-of-file occurred without reading any other character. |
| SEE ALSO | <i>fiOLib</i> |

fioRead()

fioRead()

| | |
|--------------------|--|
| NAME | <i>fioRead()</i> – read a buffer |
| SYNOPSIS | <pre>int fioRead (int fd, /* file descriptor of file to read */ char * buffer, /* buffer to receive input */ int maxbytes /* maximum number of bytes to read */)</pre> |
| DESCRIPTION | This routine repeatedly calls the routine <i>read()</i> until <i>maxbytes</i> have been read into <i>buffer</i> . If EOF is reached, the number of bytes read will be less than <i>maxbytes</i> . |
| RETURNS | The number of bytes read, or ERROR if there is an error during the read operation. |
| SEE ALSO | <i>fioLib</i> , <i>read()</i> |

floatInit()

| | |
|--------------------|---|
| NAME | <i>floatInit()</i> – initialize floating-point I/O support |
| SYNOPSIS | <pre>void floatInit (void)</pre> |
| DESCRIPTION | This routine must be called if floating-point format specifications are to be supported by the <i>printf()/scanf()</i> family of routines. If the configuration macro INCLUDE_FLOATING_POINT is defined, it is called by the root task, <i>usrRoot()</i> , in <i>usrConfig.c</i> . |
| RETURNS | N/A |
| SEE ALSO | <i>floatLib</i> |

floor()

| | |
|----------------------|--|
| NAME | <i>floor()</i> – compute the largest integer less than or equal to a specified value (ANSI) |
| SYNOPSIS | <pre>double floor (double v /* value to find the floor of */)</pre> |
| DESCRIPTION | This routine returns the largest integer less than or equal to <i>v</i> , in double precision. |
| INCLUDE FILES | math.h |
| RETURNS | The largest integral value less than or equal to <i>v</i> , in double precision. |
| SEE ALSO | ansiMath, mathALib |

floorf()

| | |
|----------------------|--|
| NAME | <i>floorf()</i> – compute the largest integer less than or equal to a specified value (ANSI) |
| SYNOPSIS | <pre>float floorf (float v /* value to find the floor of */)</pre> |
| DESCRIPTION | This routine returns the largest integer less than or equal to <i>v</i> , in single precision. |
| INCLUDE FILES | math.h |
| RETURNS | The largest integral value less than or equal to <i>v</i> , in single precision. |
| SEE ALSO | mathALib |

fmod()

fmod()

| | |
|----------------------|---|
| NAME | <i>fmod()</i> – compute the remainder of x/y (ANSI) |
| SYNOPSIS | <pre>double fmod (double x, /* numerator */ double y /* denominator */)</pre> |
| DESCRIPTION | This routine returns the remainder of x/y with the sign of x , in double precision. |
| INCLUDE FILES | math.h |
| RETURNS | The value $x - i * y$, for some integer i . If y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y . If y is zero, <i>fmod()</i> returns zero. |
| ERRNO | EDOM |
| SEE ALSO | ansiMath, mathALib |

fmodf()

| | |
|----------------------|---|
| NAME | <i>fmodf()</i> – compute the remainder of x/y (ANSI) |
| SYNOPSIS | <pre>float fmodf (float x, /* numerator */ float y /* denominator */)</pre> |
| DESCRIPTION | This routine returns the remainder of x/y with the sign of x , in single precision. |
| INCLUDE FILES | math.h |
| RETURNS | The single-precision modulus of x/y . |
| SEE ALSO | mathALib |

fnattach()

| | |
|--------------------|---|
| NAME | <i>fnattach()</i> – publish the fn network interface and initialize the driver and device |
| SYNOPSIS | <pre>STATUS fnattach (int unit /* unit number */)</pre> |
| DESCRIPTION | The routine publishes the fn interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state. |
| RETURNS | OK or ERROR. |
| SEE ALSO | <i>if_fn</i> |

fopen()

| | |
|--------------------|---|
| NAME | <i>fopen()</i> – open a file specified by name (ANSI) |
| SYNOPSIS | <pre>FILE * fopen (const char * file, /* name of file */ const char * mode /* mode */)</pre> |
| DESCRIPTION | This routine opens a file whose name is the string pointed to by <i>file</i> and associates a stream with it. The argument <i>mode</i> points to a string beginning with one of the following sequences: r open text file for reading w truncate to zero length or create text file for writing a append; open or create text file for writing at end-of-file rb open binary file for reading |

fopen()**wb**

truncate to zero length or create binary file for writing

ab

append; open or create binary file for writing at end-of-file

r+

open text file for update (reading and writing)

w+

truncate to zero length or create text file for update.

a+

append; open or create text file for update, writing at end-of-file

r+b / rb+

open binary file for update (reading and writing)

w+b / wb+

truncate to zero length or create binary file for update

a+b / ab+

append; open or create binary file for update, writing at end-of-file

Opening a file with read mode (**r** as the first character in the *mode* argument) fails if the file does not exist or cannot be read.

Opening a file with append mode (**a** as the first character in the *mode* argument) causes all subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening calls to *fseek()*. In some implementations, opening a binary file with append mode (**b** as the second or third character in the *mode* argument) may initially position the file position indicator for the stream beyond the last data written, because of null character padding. In VxWorks, whether append mode is supported is device-specific.

When a file is opened with update mode (**+** as the second or third character in the *mode* argument), both input and output may be performed on the associated stream. However, output may not be directly followed by input without an intervening call to *fflush()* or to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*), and input may not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file. Opening (or creating) a text file with update mode may instead open (or create) a binary stream in some implementations.

When opened, a stream is fully buffered if and only if it can be determined not to refer to an interactive device. The error and end-of-file indicators for the stream are cleared.

INCLUDE FILES **stdio.h**

RETURNS A pointer to the object controlling the stream, or a null pointer if the operation fails.

SEE ALSO **ansiStdio**, *fdopen()*, *freopen()*

fp()

| | |
|--------------------|--|
| NAME | <i>fp()</i> – return the contents of register fp (i960) |
| SYNOPSIS | <pre>int fp (int taskId /* task ID, 0 means default task */)</pre> |
| DESCRIPTION | This command extracts the contents of register fp , the frame pointer, from the TCB of a specified task. If <i>taskId</i> is omitted or 0, the current default task is assumed. |
| RETURNS | The contents of the fp register. |
| SEE ALSO | dbgArchLib , <i>VxWorks Programmer's Guide: Target Shell</i> |

fp0()

| | |
|--------------------|---|
| NAME | <i>fp0()</i> – return the contents of register fp0 (also fp1 – fp3) (i960KB, i960SB) |
| SYNOPSIS | <pre>double fp0 (volatile int taskId /* task ID, 0 means default task */)</pre> |
| DESCRIPTION | This command extracts the contents of the floating-point register fp0 from the TCB of a specified task. If <i>taskId</i> is omitted or 0, the current default task is assumed. Routines are provided for the floating-point registers fp0 – fp3 : <i>fp0()</i> – <i>fp3()</i> . |
| RETURNS | The contents of the fp0 register (or the requested register). |
| SEE ALSO | dbgArchLib , <i>VxWorks Programmer's Guide: Target Shell</i> |

fppInit()

fppInit()

| | |
|--------------------|---|
| NAME | <i>fppInit()</i> – initialize floating-point coprocessor support |
| SYNOPSIS | <code>void fppInit (void)</code> |
| DESCRIPTION | This routine initializes floating-point coprocessor support and must be called before using the floating-point coprocessor. This is done automatically by the root task, <i>usrRoot()</i> , in <i>usrConfig.c</i> when the configuration macro <code>INCLUDE_HW_FP</code> is defined. |
| RETURNS | N/A |
| SEE ALSO | <i>fppLib</i> |

fppProbe()

| | |
|--------------------|---|
| NAME | <i>fppProbe()</i> – probe for the presence of a floating-point coprocessor |
| SYNOPSIS | <code>STATUS fppProbe (void)</code> |
| DESCRIPTION | <p>This routine determines whether there is a floating-point coprocessor in the system. The implementation of this routine is architecture-dependent:</p> <p>MC680x0, SPARC, i386/i486: This routine sets the illegal coprocessor opcode trap vector and executes a coprocessor instruction. If the instruction causes an exception, <i>fppProbe()</i> returns <code>ERROR</code>. Note that this routine saves and restores the illegal coprocessor opcode trap vector that was there prior to this call.</p> <p>The probe is only performed the first time this routine is called. The result is stored in a static and returned on subsequent calls without actually probing.</p> <p>i960: This routine merely indicates whether VxWorks was compiled with the flag <code>-DCPU=I960KB</code>.</p> <p>MIPS This routine simply reads the R-Series status register and reports the bit that indicates whether coprocessor 1 is usable. This bit must be correctly initialized in the BSP.</p> <p>ARM This routine currently returns <code>ERROR</code> to indicate no floating-point coprocessor support.</p> |

RETURNS OK, or ERROR if there is no floating-point coprocessor.

SEE ALSO `fppArchLib`

fppRestore()

NAME *fppRestore()* – restore the floating-point coprocessor context

SYNOPSIS

```
void fppRestore
(
    FP_CONTEXT * pFpContext /* where to restore context from */
)
```

DESCRIPTION This routine restores the floating-point coprocessor context. The context restored is:

MC680x0:

- registers `fpcr`, `fpsr`, and `fpiar`
- registers `f0` – `f7`
- internal state frame (if NULL, the other registers are not saved.)

SPARC

- registers `fsr` and `fpq`
- registers `f0` – `f31`

i960:

- registers `fp0` – `fp3`

MIPS

- register `fpcsr`
- registers `fp0` – `fp31`

i386/i486:

- control word, status word, tag word, IP offset, CS selector, data operand offset, and operand selector
- registers `st0` – `st7`

ARM

- currently, on this architecture, this routine does nothing.

RETURNS N/A

SEE ALSO `fppArchLib`, `fppSave()`

fppSave()

| | |
|--------------------|---|
| NAME | <i>fppSave()</i> – save the floating-point coprocessor context |
| SYNOPSIS | <pre>void fppSave (FP_CONTEXT * pFpContext /* where to save context */)</pre> |
| DESCRIPTION | <p>This routine saves the floating-point coprocessor context. The context saved is:</p> <p>MC680x0:</p> <ul style="list-style-type: none">– registers fpcr, fpsr, and fpiar– registers f0 – f7– internal state frame (if NULL, the other registers are not saved.) <p>SPARC</p> <ul style="list-style-type: none">– registers fsr and fpq– registers f0 – f31 <p>i960:</p> <ul style="list-style-type: none">– registers fp0 – fp3 <p>MIPS</p> <ul style="list-style-type: none">– register fpcsr– registers fp0 – fp31 <p>i386/i486:</p> <ul style="list-style-type: none">– control word, status word, tag word, IP offset, CS selector, data operand offset, and operand selector (4 bytes each)– registers st0 – st7 (8 bytes each) <p>ARM</p> <ul style="list-style-type: none">– currently, on this architecture, this routine does nothing. |
| RETURNS | N/A |
| SEE ALSO | fppArchLib , <i>fppRestore()</i> |

fppShowInit()

- NAME** *fppShowInit()* – initialize the floating-point show facility
- SYNOPSIS** `void fppShowInit (void)`
- DESCRIPTION** This routine links the floating-point show facility into the VxWorks system. It is called automatically when the floating-point show facility is configured into VxWorks using either of the following methods:
- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
 - If you use the Tornado project facility, select `INCLUDE_HW_FP_SHOW`.
- RETURNS** N/A
- SEE ALSO** *fppShow*

fppTaskRegsGet()

- NAME** *fppTaskRegsGet()* – get the floating-point registers from a task TCB
- SYNOPSIS**

```
STATUS fppTaskRegsGet
(
    int          task,      /* task to get info about */
    FPREG_SET * pFpRegSet /* ptr to floating-point register set */
)
```
- DESCRIPTION** This routine copies a task's floating-point registers and/or status registers to the locations whose pointers are passed as parameters. The floating-point registers are copied into an array containing all the registers.
- NOTE** This routine only works well if *task* is not the calling task. If a task tries to discover its own registers, the values will be stale (that is, left over from the last task switch).
- RETURNS** OK, or ERROR if there is no floating-point support or there is an invalid state.
- SEE ALSO** *fppArchLib*, *fppTaskRegsSet()*

fppTaskRegsSet()

- NAME** *fppTaskRegsSet()* – set the floating-point registers of a task
- SYNOPSIS**
- ```
STATUS fppTaskRegsSet
(
 int task, /* task to set registers for */
 FPREG_SET * pFpRegSet /* ptr to floating-point register set */
)
```
- DESCRIPTION** This routine loads the specified values into the TCB of a specified task. The register values are copied from the array at *pFpRegSet*.
- RETURNS** OK, or ERROR if there is no floating-point support or there is an invalid state.
- SEE ALSO** *fppArchLib*, *fppTaskRegsGet()*

---

## ***fppTaskRegsShow()***

- NAME** *fppTaskRegsShow()* – print the contents of a task's floating-point registers
- SYNOPSIS**
- ```
void fppTaskRegsShow
(
    int task /* task to display floating point registers for */
)
```
- DESCRIPTION** This routine prints to standard output the contents of a task's floating-point registers.
- RETURNS** N/A
- SEE ALSO** *fppShow*

fprintf()

NAME *fprintf()* – write a formatted string to a stream (ANSI)

SYNOPSIS

```
int fprintf
(
    FILE *      fp, /* stream to write to */
    const char * fmt /* format string */
)
```

DESCRIPTION This routine writes output to a specified stream under control of the string *fmt*. The string *fmt* contains ordinary characters, which are written unchanged, plus conversion specifications, which cause the arguments that follow *fmt* to be converted and printed as part of the formatted string.

The number of arguments for the format is arbitrary, but they must correspond to the conversion specifications in *fmt*. If there are insufficient arguments, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The routine returns when the end of the format string is encountered.

The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: ordinary multibyte characters (not %) that are copied unchanged to the output stream; and conversion specification, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the % character. After the %, the following appear in sequence:

- Zero or more flags (in any order) that modify the meaning of the conversion specification.
- An optional minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces (by default) on the left (or right, if the left adjustment flag, described later, has been given) to the field width. The field width takes the form of an asterisk (*) (described later) or a decimal integer.
- An optional precision that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, and **X** conversions, the number of digits to appear after the decimal-point character for **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversions, or the maximum number of characters to be written from a string in the **s** conversion. The precision takes the form of a period (.) followed either by an asterisk (*) (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional **h** specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **short int** or **unsigned short int** argument (the argument will have been

fprintf()

promoted according to the integral promotions, and its value converted to **short int** or **unsigned short int** before printing); an optional **h** specifying that a following **n** conversion specifier applies to a pointer to a **short int** argument; an optional **l** (el) specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **long int** or **unsigned long int** argument; or an optional **L** specifying that a following **n** conversion specifier applies to a pointer to a **long int** argument. If an **h** or **L** appears with any other conversion specifier, the behavior is undefined.

WARNING

ANSI C also specifies an optional **L** in some of the same contexts as **l** above, corresponding to a **long double** argument. However, the current release of the VxWorks libraries does not support **long double** data; using the optional **L** gives unpredictable results.

- A character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both, can be indicated by an asterisk (*). In this case, an **int** argument supplies the field width or precision. The arguments specifying field width, or precision, or both, should appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field. (it will be right-justified if this flag is not specified.)
- + The result of a signed conversion will always begin with a plus or minus sign. (It will begin with a sign only when a negative value is converted if this flag is not specified.)

space

If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space will be prefixed to the result. If the **space** and **+** flags both appear, the **space** flag will be ignored.

#

The result is to be converted to an "alternate form." For **o** conversion it increases the precision to force the first digit of the result to be a zero. For **x** (or **X**) conversion, a non-zero result will have "0x" (or "0X") prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if no digit follows it). For **g** and **G** conversions, trailing zeros will not be removed from the result. For other conversions, the behavior is undefined.

0

For **d**, **i**, **o**, **u**, **x**, **X**, **e**, **E**, **f**, **g**, and **G** conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is

performed. If the **0** and **-** flags both appear, the **0** flag will be ignored. For **d**, **i**, **o**, **u**, **x**, and **X** conversions, if a precision is specified, the **0** flag will be ignored. For other conversions, the behavior is undefined.

The conversion specifiers and their meanings are:

d, i

The **int** argument is converted to signed decimal in the style **[-]dddd**. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

o, u, x, X

The **unsigned int** argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x** or **X**) in the style **dddd**; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

f

The **double** argument is converted to decimal notation in the style **[-]ddd.ddd**, where the number of digits after the decimal point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the **#** flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.

e, E

The **double** argument is converted in the style **[-]d.ddde+/-dd**, where there is one digit before the decimal-point character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the **#** flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The **E** conversion specifier will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

g, G

The **double** argument is converted in style **f** or **e** (or in style **E** in the case of a **G** conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as 1. The style used depends on the value converted; style **e** (or **E**) will be used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a decimal-point character appears only if it is followed by a digit.

fprintf()

c

The **int** argument is converted to an **unsigned char**, and the resulting character is written.

s

The argument should be a pointer to an array of character type. Characters from the array are written up to (but not including) a terminating null character; if the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array will contain a null character.

p

The argument should be a pointer to **void**. The value of the pointer is converted to a sequence of printable characters, in hexadecimal representation (prefixed with "0x").

n

The argument should be a pointer to an integer into which the number of characters written to the output stream so far by this call to *fprintf()* is written. No argument is converted.

%

A % is written. No argument is converted. The complete conversion specification is %%.
If a conversion specification is invalid, the behavior is undefined.

If any argument is, or points to, a union or an aggregate (except for an array of character type using **s** conversion, or a pointer using **p** conversion), the behavior is undefined.

In no case does a non-existent or small field width cause truncation of a field if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

INCLUDE FILES

stdio.h

RETURNS

The number of characters written, or a negative value if an output error occurs.

SEE ALSO

ansiStdio, printf()

fputc()

| | |
|----------------------|--|
| NAME | <i>fputc()</i> – write a character to a stream (ANSI) |
| SYNOPSIS | <pre>int fputc (int c, /* character to write */ FILE * fp /* stream to write to */)</pre> |
| DESCRIPTION | <p>This routine writes a character <i>c</i> to a specified stream, at the position indicated by the stream's file position indicator (if defined), and advances the indicator appropriately.</p> <p>If the file cannot support positioning requests, or if the stream was opened in append mode, the character is appended to the output stream.</p> |
| INCLUDE FILES | <code>stdio.h</code> |
| RETURNS | The character written, or EOF if a write error occurs, with the error indicator set for the stream. |
| SEE ALSO | <code>ansiStdio</code> , <i>fputs()</i> , <i>putc()</i> |

fputs()

| | |
|----------------------|--|
| NAME | <i>fputs()</i> – write a string to a stream (ANSI) |
| SYNOPSIS | <pre>int fputs (const char * s, /* string */ FILE * fp /* stream to write to */)</pre> |
| DESCRIPTION | <p>This routine writes the string <i>s</i>, minus the terminating NULL character, to a specified stream.</p> |
| INCLUDE FILES | <code>stdio.h</code> |
| RETURNS | A non-negative value, or EOF if a write error occurs. |
| SEE ALSO | <code>ansiStdio</code> , <i>fputc()</i> |

fread()

fread()

| | |
|----------------------|--|
| NAME | <i>fread()</i> – read data into an array (ANSI) |
| SYNOPSIS | <pre>int fread (void * buf, /* where to copy data */ size_t size, /* element size */ size_t count, /* no. of elements */ FILE * fp /* stream to read from */)</pre> |
| DESCRIPTION | This routine reads, into the array <i>buf</i> , up to <i>count</i> elements of size <i>size</i> , from a specified stream <i>fp</i> . The file position indicator for the stream (if defined) is advanced by the number of characters successfully read. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate. If a partial element is read, its value is indeterminate. |
| INCLUDE FILES | stdio.h |
| RETURNS | The number of elements successfully read, which may be less than <i>count</i> if a read error or end-of-file is encountered; or zero if <i>size</i> or <i>count</i> is zero, with the contents of the array and the state of the stream remaining unchanged. |
| SEE ALSO | ansiStdio |

free()

| | |
|--------------------|---|
| NAME | <i>free()</i> – free a block of memory (ANSI) |
| SYNOPSIS | <pre>void free (void * ptr /* pointer to block of memory to free */)</pre> |
| DESCRIPTION | This routine returns to the free memory pool a block of memory previously allocated with <i>malloc()</i> or <i>calloc()</i> . |
| RETURNS | N/A |

SEE ALSO *memPartLib*, *malloc()*, *calloc()*, *American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: General Utilities (stdlib.h)*

freopen()

NAME *freopen()* – open a file specified by name (ANSI)

SYNOPSIS

```
FILE * freopen
(
    const char * file, /* name of file */
    const char * mode, /* mode */
    FILE *      fp    /* stream */
)
```

DESCRIPTION This routine opens a file whose name is the string pointed to by *file* and associates it with a specified stream *fp*. The *mode* argument is used just as in the *fopen()* function.

This routine first attempts to close any file that is associated with the specified stream. Failure to close the file successfully is ignored. The error and end-of-file indicators for the stream are cleared.

Typically, *freopen()* is used to attach the already-open streams *stdin*, *stdout*, and *stderr* to other files.

INCLUDE FILES *stdio.h*

RETURNS The value of *fp*, or a null pointer if the open operation fails.

SEE ALSO *ansiStdio*, *fopen()*

frexp()

NAME *frexp()* – break a floating-point number into a normalized fraction and power of 2 (ANSI)

SYNOPSIS

```
double frexp
(
    double value, /* number to be normalized */
    int *  pexp  /* pointer to the exponent */
)
```

fscanf()

| | |
|----------------------|--|
| DESCRIPTION | This routine breaks a double-precision number <i>value</i> into a normalized fraction and integral power of 2. It stores the integer exponent in <i>pexp</i> . |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision value <i>x</i> , such that the magnitude of <i>x</i> is in the interval [1/2,1] or zero, and <i>value</i> equals <i>x</i> times 2 to the power of <i>pexp</i> . If <i>value</i> is zero, both parts of the result are zero. |
| ERRNO | EDOM |
| SEE ALSO | ansiMath |

fscanf()

| | |
|--------------------|---|
| NAME | <i>fscanf()</i> – read and convert characters from a stream (ANSI) |
| SYNOPSIS | <pre>int fscanf (FILE * fp, /* stream to read from */ char const * fmt /* format string */) </pre> |
| DESCRIPTION | <p>This routine reads characters from a specified stream, and interprets them according to format specifications in the string <i>fmt</i>, which specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.</p> <p>If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.</p> <p>The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: one or more white-space characters; an ordinary multibyte character (neither % nor a white-space character); or a conversion specification. Each conversion specification is introduced by the % character. After the %, the following appear in sequence:</p> <ul style="list-style-type: none"> – An optional assignment-suppressing character *. – An optional non-zero decimal integer that specifies the maximum field width. – An optional h or l (el) indicating the size of the receiving object. The conversion specifiers d, i, and n should be preceded by h if the corresponding argument is a pointer to short int rather than a pointer to int, or by l if it is a pointer to long int. |

Similarly, the conversion specifiers **o**, **u**, and **x** shall be preceded by **h** if the corresponding argument is a pointer to **unsigned short int** rather than a pointer to **unsigned int**, or by **l** if it is a pointer to **unsigned long int**. Finally, the conversion specifiers **e**, **f**, and **g** shall be preceded by **l** if the corresponding argument is a pointer to **double** rather than a pointer to **float**. If an **h** or **l** appears with any other conversion specifier, the behavior is undefined.

- WARNING: ANSI C also specifies an optional **L** in some of the same contexts as **l** above, corresponding to a **long double *** argument. However, the current release of the VxWorks libraries does not support **long double** data; using the optional **L** gives unpredictable results.
- A character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The *fscanf()* routine executes each directive of the format in turn. If a directive fails, as detailed below, *fscanf()* returns. Failures are described as input failures (due to the unavailability of input characters), or matching failures (due to inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary multibyte character is executed by reading the next characters of the stream. If one of the characters differs from one comprising the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, described below for each specifier. A conversion specification is executed in the following steps:

Input white-space characters (as specified by the *isspace()* function) are skipped, unless the specification includes a **[**, **c**, or **n** specifier.

An input item is read from the stream, unless an **n** specifier is included. An input item is defined as the longest matching sequence of input characters, unless it exceeds a specified field width, in which case it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails: this condition is a matching failure, unless an error prevented input from the stream, in which case it is an input failure.

Except in the case of a **%** specifier, the input item is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a *****, the result of the conversion is placed in the object pointed to by the first argument following the *fmt* argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The following conversion specifiers are valid:

fscanf()

d

Matches an optionally signed decimal integer whose format is the same as expected for the subject sequence of the *strtol()* function with the value 10 for the *base* argument. The corresponding argument should be a pointer to **int**.

i

Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the *strtol()* function with the value 0 for the *base* argument. The corresponding argument should be a pointer to **int**.

o

Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the *strtoul()* function with the value 8 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

u

Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtoul()* function with the value 10 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

x

Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the *strtoul()* function with the value 16 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

e, f, g

Match an optionally signed floating-point number, whose format is the same as expected for the subject string of the *strtod()* function. The corresponding argument should be a pointer to **float**.

s

Matches a sequence of non-white-space characters. The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which will be added automatically.

[

Matches a non-empty sequence of characters from a set of expected characters (the **scanset**). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which is added automatically. The conversion specifier includes all subsequent character in the format string, up to and including the matching right bracket (**]**). The characters between the brackets (the **scanlist**) comprise the scanset, unless the character after the left bracket is a circumflex (**^**) in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with "**[]**" or "**[^]**", the right bracket character is in the scanlist and the next right bracket character is the matching right bracket that ends the specification; otherwise the first right bracket character is the one that ends the specification.

c Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added.

p Matches an implementation-defined set of sequences, which should be the same as the set of sequences that may be produced by the %p conversion of the *fprintf()* function. The corresponding argument should be a pointer to a pointer to **void**. VxWorks defines its pointer input field to be consistent with pointers written by the *fprintf()* function ("0x" hexadecimal notation). If the input item is a value converted earlier during the same program execution, the pointer that results should compare equal to that value; otherwise the behavior of the %p conversion is undefined.

n No input is consumed. The corresponding argument should be a pointer to **int** into which the number of characters read from the input stream so far by this call to *fscanf()* is written. Execution of a %n directive does not increment the assignment count returned when *fscanf()* completes execution.

% Matches a single %; no conversion or assignment occurs. The complete conversion specification is %%.
If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers **E**, **G**, and **X** are also valid and behave the same as **e**, **g**, and **x**, respectively.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the %n directive.

INCLUDE FILES **stdio.h**

RETURNS The number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure; or EOF if an input failure occurs before any conversion.

SEE ALSO **ansiStdio**, *scanf()*, *sscanf()*

fseek()

fseek()

NAME *fseek()* – set the file position indicator for a stream (ANSI)

SYNOPSIS

```
int fseek
(
    FILE * fp,      /* stream */
    long  offset,  /* offset from whence */
    int   whence  /* position to offset from: */
                  /* SEEK_SET = beginning SEEK_CUR */
                  /* current position SEEK_END = end-of-file */
)
```

DESCRIPTION This routine sets the file position indicator for a specified stream. For a binary stream, the new position, measured in characters from the beginning of the file, is obtained by adding *offset* to the position specified by *whence*, whose possible values are:

SEEK_SET
the beginning of the file.

SEEK_CUR
the current value of the file position indicator.

SEEK_END
the end of the file.

A binary stream does not meaningfully support *fseek()* calls with a *whence* value of **SEEK_END**.

For a text stream, either *offset* is zero, or *offset* is a value returned by an earlier call to *ftell()* on the stream, in which case *whence* should be **SEEK_SET**.

A successful call to *fseek()* clears the end-of-file indicator for the stream and undoes any effects of *ungetc()* on the same stream. After an *fseek()* call, the next operation on an update stream can be either input or output.

INCLUDE FILES **stdio.h**

RETURNS Non-zero only for a request that cannot be satisfied.

ERRNO **EINVAL**

SEE ALSO *ansiStdio*, *ftell()*

fsetpos()

- NAME** *fsetpos()* – set the file position indicator for a stream (ANSI)
- SYNOPSIS**
- ```
int fsetpos
(
 FILE * iop, /* stream */
 const fpos_t * pos /* position, obtained by fgetpos() */
)
```
- DESCRIPTION**
- This routine sets the file position indicator for a specified stream *iop* according to the value of the object pointed to by *pos*, which is a value obtained from an earlier call to *fgetpos()* on the same stream.
- A successful call to *fsetpos()* clears the end-of-file indicator for the stream and undoes any effects of *ungetc()* on the same stream. After an *fsetpos()* call, the next operation on an update stream may be either input or output.
- INCLUDE FILES** `stdio.h`
- RETURNS** Zero, or non-zero if the call fails, with **errno** set to indicate the error.
- SEE ALSO** `ansiStdio`, *fgetpos()*

---

## *fsrShow()*

- NAME** *fsrShow()* – display the meaning of a specified fsr value, symbolically (SPARC)
- SYNOPSIS**
- ```
void fsrShow
(
    UINT fsrValue /* fsr value to show */
)
```
- DESCRIPTION**
- This routine displays the meaning of all the fields in a specified **fsr** value, symbolically. Extracted from `reg.h`:

Definition of bits in the Sun-4 FSR (Floating-point Status Register)

```
-----
| RD | RP | TEM | res | FTT | QNE | PR | FCC | AEXC | CEXC |
|----|----|-----|-----|-----|-----|----|-----|-----|-----|
| 31 30 29 28 27 23 22 17 16 14 13 12 11 10 9 5 4 0
```

fstat()

For compatibility with future revisions, reserved bits are defined to be initialized to zero and, if written, must be preserved.

EXAMPLE

```
-> fsrShow 0x12345678
Rounding Direction: nearest or even if tie.
Rounding Precision: single.
Trap Enable Mask:
    underflow.
Floating-point Trap Type: IEEE exception.
Queue Not Empty: FALSE;
Partial Remainder: TRUE;
Condition Codes: less than.
Accumulated exceptions:
    inexact divide-by-zero invalid.
Current exceptions:
    overflow invalid
```

RETURNS N/A

SEE ALSO **dbgArchLib**, *SPARC Architecture Manual*

fstat()

NAME *fstat()* – get file status information (POSIX)

SYNOPSIS

```
STATUS fstat
(
    int          fd, /* file descriptor for file to check */
    struct stat * pStat /* pointer to stat structure */
)
```

DESCRIPTION This routine obtains various characteristics of a file (or directory). The file must already have been opened using *open()* or *creat()*. The *fd* parameter is the file descriptor returned by *open()* or *creat()*.

The *pStat* parameter is a pointer to a **stat** structure (defined in **stat.h**). This structure must be allocated before *fstat()* is called.

On return, the **stat** structure fields are updated to reflect the characteristics of the file.

RETURNS OK or ERROR.

SEE ALSO **dirLib**, *stat()*, *ls()*

fstatfs()

| | |
|--------------------|--|
| NAME | <i>fstatfs()</i> – get file status information (POSIX) |
| SYNOPSIS | <pre>STATUS fstatfs (int fd, /* file descriptor for file to check */ struct statfs * pStat /* pointer to statfs structure */)</pre> |
| DESCRIPTION | <p>This routine obtains various characteristics of a file system. A file in the file system must already have been opened using <i>open()</i> or <i>creat()</i>. The <i>fd</i> parameter is the file descriptor returned by <i>open()</i> or <i>creat()</i>.</p> <p>The <i>pStat</i> parameter is a pointer to a stat structure (defined in stat.h). This structure must be allocated before <i>fstat()</i> is called.</p> <p>Upon return, the fields in the statfs structure are updated to reflect the characteristics of the file.</p> |
| RETURNS | OK or ERROR. |
| SEE ALSO | <i>dirLib</i> , <i>statfs()</i> , <i>ls()</i> |

ftell()

| | |
|----------------------|--|
| NAME | <i>ftell()</i> – return the current value of the file position indicator for a stream (ANSI) |
| SYNOPSIS | <pre>long ftell (FILE * fp /* stream */)</pre> |
| DESCRIPTION | <p>This routine returns the current value of the file position indicator for a specified stream. For a binary stream, the value is the number of characters from the beginning of the file. For a text stream, the file position indicator contains unspecified information, usable by <i>fseek()</i> for returning the file position indicator to its position at the time of the <i>ftell()</i> call; the difference between two such return values is not necessarily a meaningful measure of the number of characters written or read.</p> |
| INCLUDE FILES | stdio.h |

ftpCommand()

- RETURNS** The current value of the file position indicator, or -1L if unsuccessful, with **errno** set to indicate the error.
- SEE ALSO** *ansiStdio*, *fseek()*

ftpCommand()

NAME *ftpCommand()* – send an FTP command and get the reply

SYNOPSIS

```
int ftpCommand
(
    int    ctrlSock, /* fd of control connection socket */
    char * fmt,      /* format string of command to send */
    int    arg1,     /* first of six args to format string */
    int    arg2,
    int    arg3,
    int    arg4,
    int    arg5,
    int    arg6
)
```

DESCRIPTION This routine sends the specified command on the specified socket, which should be a control connection to a remote FTP server. The command is specified as a string in *printf()* format with up to six arguments.

After the command is sent, *ftpCommand()* waits for the reply from the remote server. The FTP reply code is returned in the same way as in *ftpReplyGet()*.

EXAMPLE

```
ftpCommand (ctrlSock, "TYPE I", 0, 0, 0, 0, 0, 0); /* image-type xfer */
ftpCommand (ctrlSock, "STOR %s", file, 0, 0, 0, 0, 0); /* init file write */
```

RETURNS

- 1 = FTP_PRELIM (positive preliminary)
- 2 = FTP_COMPLETE (positive completion)
- 3 = FTP_CONTINUE (positive intermediate)
- 4 = FTP_TRANSIENT (transient negative completion)
- 5 = FTP_ERROR (permanent negative completion)

ERROR if there is a read/write error or an unexpected EOF.

SEE ALSO *ftpLib*, *ftpReplyGet()*

ftpDataConnGet()

NAME *ftpDataConnGet()* – get a completed FTP data connection

SYNOPSIS

```
int ftpDataConnGet
(
    int dataSock /* fd of data socket on which to await connection */
)
```

DESCRIPTION This routine completes a data connection initiated by a call to *ftpDataConnInit()*. It waits for a connection on the specified socket from the remote FTP server. The specified socket should be the one returned by *ftpDataConnInit()*. The connection is established on a new socket, whose file descriptor is returned as the result of this function. The original socket, specified in the argument to this routine, is closed.

Usually this routine is called after *ftpDataConnInit()* and *ftpCommand()* to initiate a data transfer from/to the remote FTP server.

RETURNS The file descriptor of the new data socket, or ERROR if the connection failed.

SEE ALSO *ftpLib*, *ftpDataConnInit()*, *ftpCommand()*

ftpDataConnInit()

NAME *ftpDataConnInit()* – initialize an FTP data connection

SYNOPSIS

```
int ftpDataConnInit
(
    int ctrlSock /* fd of associated control socket */
)
```

DESCRIPTION This routine sets up the client side of a data connection for the specified control connection. It creates the data port, informs the remote FTP server of the data port address, and listens on that data port. The server will then connect to this data port in response to a subsequent data-transfer command sent on the control connection (see the manual entry for *ftpCommand()*).

This routine must be called *before* the data-transfer command is sent; otherwise, the server's connect may fail.

ftpdDelete()

This routine is called after *ftpHookup()* and *ftpLogin()* to establish a connection with a remote FTP server at the lowest level. (For a higher-level interaction with a remote FTP server, see *ftpXfer()*.)

RETURNS The file descriptor of the data socket created, or ERROR.

SEE ALSO *ftpLib*, *ftpHookup()*, *ftpLogin()*, *ftpCommand()*, *ftpXfer()*

ftpdDelete()

NAME *ftpdDelete()* – terminate the FTP server task

SYNOPSIS `STATUS ftpdDelete (void)`

DESCRIPTION This routine halts the FTP server and closes the control connection. All client sessions are removed after completing any commands in progress. When this routine executes, no further client connections will be accepted until the server is restarted. This routine is not reentrant and must not be called from interrupt level.

NOTE If any file transfer operations are in progress when this routine is executed, the transfers will be aborted, possibly leaving incomplete files on the destination host.

RETURNS OK if shutdown completed, or ERROR otherwise.

ERRNO N/A

SEE ALSO *ftpdLib*

ftpdInit()

NAME *ftpdInit()* – initialize the FTP server task

SYNOPSIS `STATUS ftpdInit`

```
(
    FUNCPTR pLoginRtn, /* user verification routine, or NULL */
    int     stackSize /* task stack size, or 0 for default */
)
```

| | |
|--------------------|--|
| DESCRIPTION | This routine installs the password verification routine indicated by <i>pLoginRtn</i> and establishes a control connection for the primary FTP server task, which it then creates. It is called automatically during system startup if <code>INCLUDE_FTP_SERVER</code> is defined. The primary server task supports simultaneous client sessions, up to the limit specified by the global variable <code>ftpsMaxClients</code> . The default value allows a maximum of four simultaneous connections. The <i>stackSize</i> argument specifies the stack size for the primary server task. It is set to the value specified in the <code>ftpdWorkTaskStackSize</code> global variable by default. |
| RETURNS | OK if server started, or ERROR otherwise. |
| ERRNO | N/A |
| SEE ALSO | <code>ftpdLib</code> |

ftpHookup()

| | |
|--------------------|---|
| NAME | <i>ftpHookup()</i> – get a control connection to the FTP server on a specified host |
| SYNOPSIS | <pre>int ftpHookup (char * host /* server host name or inet address */)</pre> |
| DESCRIPTION | This routine establishes a control connection to the FTP server on the specified host. This is the first step in interacting with a remote FTP server at the lowest level. (For a higher-level interaction with a remote FTP server, see the manual entry for <i>ftpXfer()</i> .) |
| RETURNS | The file descriptor of the control socket, or ERROR if the Internet address or the host name is invalid, if a socket could not be created, or if a connection could not be made. |
| SEE ALSO | <code>ftpLib</code> , <i>ftpLogin()</i> , <i>ftpXfer()</i> |

ftpLogin()

NAME *ftpLogin()* – log in to a remote FTP server

SYNOPSIS

```
STATUS ftpLogin
(
    int    ctrlSock, /* fd of login control socket */
    char * user,     /* user name for host login */
    char * passwd,   /* password for host login */
    char * account   /* account for host login */
)
```

DESCRIPTION This routine logs in to a remote server with the specified user name, password, and account name, as required by the specific remote host. This is typically the next step after calling *ftpHookup()* in interacting with a remote FTP server at the lowest level. (For a higher-level interaction with a remote FTP server, see the manual entry for *ftpXfer()*).

RETURNS OK, or ERROR if the routine is unable to log in.

SEE ALSO *ftpLib*, *ftpHookup()*, *ftpXfer()*

ftpLs()

NAME *ftpLs()* – list directory contents via FTP

SYNOPSIS

```
STATUS ftpLs
(
    char * dirName /* name of directory to list */
)
```

DESCRIPTION This routine lists the contents of a directory. The content list is obtained via an NLST FTP transaction.

The local device name must be the same as the remote host name with a colon ":" as a suffix. (For example "wrs:" is the device name for the "wrs" host.)

RETURNS OK, or ERROR if could not open directory.

SEE ALSO *ftpLib*

ftpReplyGet()

NAME *ftpReplyGet()* – get an FTP command reply

SYNOPSIS

```
int ftpReplyGet
(
    int ctrlSock, /* control socket fd of FTP connection */
    BOOL expecteof /* TRUE = EOF expected, FALSE = EOF is error */
)
```

DESCRIPTION This routine gets a command reply on the specified control socket. All the lines of a reply are read (multi-line replies are indicated with the continuation character "-" as the fourth character of all but the last line).

The three-digit reply code from the first line is saved and interpreted. The left-most digit of the reply code identifies the type of code (see RETURNS below).

The caller's error status is set to the complete three-digit reply code (see the manual entry for *errnoGet()*). If the reply code indicates an error, the entire reply is printed on standard error.

If an EOF is encountered on the specified control socket, but no EOF was expected (*expecteof* == FALSE), then ERROR is returned.

RETURNS

- 1 = FTP_PRELIM (positive preliminary)
- 2 = FTP_COMPLETE (positive completion)
- 3 = FTP_CONTINUE (positive intermediate)
- 4 = FTP_TRANSIENT (transient negative completion)
- 5 = FTP_ERROR (permanent negative completion)

ERROR if there is a read/write error or an unexpected EOF.

SEE ALSO *ftpLib*

ftpXfer()

NAME *ftpXfer()* – initiate a transfer via FTP

SYNOPSIS

```
STATUS ftpXfer
(
    char * host,      /* name of server host */
    char * user,     /* user name for host login */
)
```

```
char * passwd, /* password for host login */
char * acct, /* account for host login */
char * cmd, /* command to send to host */
char * dirname, /* directory to cd to before sending command */
char * filename, /* filename to send with command */
int * pCtrlSock, /* where to return control socket fd */
int * pDataSock /* where to return data socket fd, */
/* (NULL == don't open connection) */
)
```

DESCRIPTION

This routine initiates a transfer via a remote FTP server in the following order:

- (1) Establishes a connection to the FTP server on the specified host.
- (2) Logs in with the specified user name, password, and account, as necessary for the particular host.
- (3) Sets the transfer type to image by sending the command "TYPE I".
- (4) Changes to the specified directory by sending the command "CWD *dirname*".
- (5) Sends the specified transfer command with the specified filename as an argument, and establishes a data connection. Typical transfer commands are "STOR %s", to write to a remote file, or "RETR %s", to read a remote file.

The resulting control and data connection file descriptors are returned via *pCtrlSock* and *pDataSock*, respectively.

After calling this routine, the data can be read or written to the remote server by reading or writing on the file descriptor returned in *pDataSock*. When all incoming data has been read (as indicated by an EOF when reading the data socket) and/or all outgoing data has been written, the data socket fd should be closed. The routine *ftpReplyGet()* should then be called to receive the final reply on the control socket, after which the control socket should be closed.

If the FTP command does not involve data transfer, *pDataSock* should be NULL, in which case no data connection will be established. The only FTP commands supported for this case are DELE, RMD, and MKD.

EXAMPLE

The following code fragment reads the file "/usr/fred/myfile" from the host "server", logged in as user "fred", with password "magic" and no account name.

```
#include "vxWorks.h"
#include "ftpLib.h"
int ctrlSock;
int dataSock;
char buf [512];
int nBytes;
STATUS status;
if (ftpXfer ("server", "fred", "magic", "",
```



```

        "RETR %s", "/usr/fred", "myfile",
        &ctrlSock, &dataSock) == ERROR)
    return (ERROR);
while ((nBytes = read (dataSock, buf, sizeof (buf))) > 0)
{
    ...
}
close (dataSock);
if (nBytes < 0)                /* read error? */
    status = ERROR;
if (ftpReplyGet (ctrlSock, TRUE) != FTP_COMPLETE)
    status = ERROR;
if (ftpCommand (ctrlSock, "QUIT", 0, 0, 0, 0, 0, 0) != FTP_COMPLETE)
    status = ERROR;
close (ctrlSock);

```

RETURNS OK, or ERROR if any socket cannot be created or if a connection cannot be made.

SEE ALSO *ftpLib*, *ftpReplyGet()*

ftruncate()

NAME *ftruncate()* – truncate a file (POSIX)

SYNOPSIS

```

int ftruncate
(
    int  fildes, /* fd of file to truncate */
    off_t length /* length to truncate file */
)

```

DESCRIPTION This routine truncates a file to a specified size.

RETURNS 0 (OK) or -1 (ERROR) if unable to truncate file.

ERRNO

- EROFS**
 - File resides on a read-only file system.
- EBADF**
 - File is open for reading only.
- EINVAL**
 - File descriptor refers to a file on which this operation is impossible.

SEE ALSO *ftruncate*

fwrite()

fwrite()**NAME** *fwrite()* – write from a specified array (ANSI)

SYNOPSIS

```
int fwrite
(
    const void * buf, /* where to copy from */
    size_t      size, /* element size */
    size_t      count, /* no. of elements */
    FILE *      fp    /* stream to write to */
)
```

DESCRIPTION This routine writes, from the array *buf*, up to *count* elements whose size is *size*, to a specified stream. The file position indicator for the stream (if defined) is advanced by the number of characters successfully written. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.

INCLUDE FILES `stdio.h`

RETURNS The number of elements successfully written, which will be less than *count* only if a write error is encountered.

SEE ALSO `ansiStdio`

g0()**NAME** *g0()* – return the contents of register **g0**, also **g1 – g7** (SPARC) and **g1 – g14** (i960)

SYNOPSIS

```
int g0
(
    int taskId /* task ID, 0 means default task */
)
```

DESCRIPTION This command extracts the contents of global register **g0** from the TCB of a specified task. If *taskId* is omitted or 0, the current default task is assumed.

Routines are provided for all global registers:

```
SPARC      g0() – g7()   (g0 – g7)
i960:     g0() – g14()  (g0 – g14)
```

RETURNS The contents of register **g0** (or the requested register).

SEE ALSO **dbgArchLib**, *VxWorks Programmer's Guide: Target Shell*

getc()

NAME *getc()* – return the next character from a stream (ANSI)

SYNOPSIS

```
int getc
(
    FILE * fp /* input stream */
)
```

DESCRIPTION This routine is equivalent to *fgetc()*, except that if it is implemented as a macro, it may evaluate *fp* more than once; thus the argument should never be an expression with side effects.

If the stream is at end-of-file, the end-of-file indicator for the stream is set; if a read error occurs, the error indicator is set.

INCLUDE FILES **stdio.h**

RETURNS The next character from the stream, or EOF if the stream is at end-of-file or a read error occurs.

SEE ALSO **ansiStdio**, *fgetc()*

getchar()

NAME *getchar()* – return the next character from the standard input stream (ANSI)

SYNOPSIS **int getchar (void)**

DESCRIPTION This routine returns the next character from the standard input stream and advances the file position indicator.

It is equivalent to *getc()* with the stream argument **stdin**.

If the stream is at end-of-file, the end-of-file indicator is set; if a read error occurs, the error indicator is set.

getcwd()

| | |
|----------------------|---|
| INCLUDE FILES | stdio.h |
| RETURNS | The next character from the standard input stream, or EOF if the stream is at end-of-file or a read error occurs. |
| SEE ALSO | ansiStdio, getc(), fgetc() |

getcwd()

| | |
|--------------------|--|
| NAME | <i>getcwd()</i> – get the current default path (POSIX) |
| SYNOPSIS | <pre>char *getcwd (char * buffer, /* where to return the pathname */ int size /* size in bytes of buffer */)</pre> |
| DESCRIPTION | This routine copies the name of the current default path to <i>buffer</i> . It provides the same functionality as <i>ioDefPathGet()</i> and is provided for POSIX compatibility. |
| RETURNS | A pointer to the supplied buffer, or NULL if <i>size</i> is too small to hold the current default path. |
| SEE ALSO | ioLib, ioDefPathSet(), ioDefPathGet(), chdir() |

getenv()

| | |
|--------------------|--|
| NAME | <i>getenv()</i> – get an environment variable (ANSI) |
| SYNOPSIS | <pre>char *getenv (const char * name /* env variable to get value for */)</pre> |
| DESCRIPTION | This routine searches the environment list (see the UNIX BSD 4.3 manual entry for environ(5V)) for a string of the form "name=value" and returns the value portion of the string, if the string is present; otherwise it returns a NULL pointer. |
| RETURNS | A pointer to the string value, or a NULL pointer. |

SEE ALSO *envLibInit()*, *putenv()*, UNIX BSD 4.3 manual entry for **environ(5V)**, *American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: General Utilities (stdlib.h)*

gethostname()

NAME *gethostname()* – get the symbolic name of this machine

SYNOPSIS

```
int gethostname
(
    char * name,    /* machine name */
    int  nameLen /* length of name */
)
```

DESCRIPTION This routine gets the target machine's symbolic name, which can be used for identification.

RETURNS OK or ERROR.

SEE ALSO **hostLib**

getpeername()

NAME *getpeername()* – get the name of a connected peer

SYNOPSIS

```
STATUS getpeername
(
    int          s,          /* socket descriptor */
    struct sockaddr * name, /* where to put name */
    int *        namelen /* space available in name, later filled in */
                        /* actual name size */
)
```

DESCRIPTION This routine gets the name of the peer connected to socket *s*. The parameter *namelen* should be initialized to indicate the amount of space referenced by *name*. On return, the name of the socket is copied to *name* and the size of the socket name is copied to *namelen*.

RETURNS OK, or ERROR if the socket is invalid or not connected.

SEE ALSO **sockLib**

gets()

gets()

| | |
|----------------------|---|
| NAME | <i>gets()</i> – read characters from the standard input stream (ANSI) |
| SYNOPSIS | <pre>char * gets (char * buf /* output array */)</pre> |
| DESCRIPTION | <p>This routine reads characters from the standard input stream into the array <i>buf</i> until end-of-file is encountered or a new-line is read. Any new-line character is discarded, and a null character is written immediately after the last character read into the array.</p> <p>If end-of-file is encountered and no characters have been read, the contents of the array remain unchanged. If a read error occurs, the array contents are indeterminate.</p> |
| INCLUDE FILES | stdio.h |
| RETURNS | A pointer to <i>buf</i> , or a null pointer if (1) end-of-file is encountered and no characters have been read, or (2) there is a read error. |
| SEE ALSO | ansiStdio |

getsockname()

| | |
|--------------------|--|
| NAME | <i>getsockname()</i> – get a socket name |
| SYNOPSIS | <pre>STATUS getsockname (int s, /* socket descriptor */ struct sockaddr * name, /* where to return name */ int * namelen /* space available in name, later filled in */ /* actual name size */)</pre> |
| DESCRIPTION | <p>This routine gets the current name for the specified socket <i>s</i>. The parameter <i>namelen</i> should be initialized to indicate the amount of space referenced by <i>name</i>. On return, the name of the socket is copied to <i>name</i> and the size of the socket name is copied to <i>namelen</i>.</p> |
| RETURNS | OK, or ERROR if the socket is invalid or not connected. |
| SEE ALSO | sockLib |

getsockopt()

- NAME** `getsockopt()` – get socket options
- SYNOPSIS**
- ```
STATUS getsockopt
(
 int s, /* socket */
 int level, /* protocol level for options */
 int optname, /* name of option */
 char * optval, /* where to put option */
 int * optlen /* where to put option length */
)
```
- DESCRIPTION**
- This routine returns relevant option values associated with a socket. To manipulate options at the "socket" level, *level* should be `SOL_SOCKET`. Any other levels should use the appropriate protocol number. The parameter *optlen* should be initialized to indicate the amount of space referenced by *optval*. On return, the value of the option is copied to *optval* and the actual size of the option is copied to *optlen*.
- Although *optval* is passed as a `char *`, the actual variable whose address gets passed in should be an integer or a structure, depending on which *optname* is being passed. Refer to `setsockopt()` to determine the correct type of the actual variable (whose address should then be cast to a `char *`).
- RETURNS** OK, or ERROR if there is an invalid socket, an unknown option, or the call is unable to get the specified option.
- EXAMPLE**
- Because `SO_REUSEADDR` has an integer parameter, the variable to be passed to `getsockopt()` should be declared as
- ```
int reuseVal;
```
- and passed in as
- ```
(char *)&reuseVal.
```
- Otherwise the user might mistakenly declare `reuseVal` as a character, in which case `getsockopt()` will only return the first byte of the integer representing the state of this option. Then whether the return value is correct or always 0 depends on the endianness of the machine.
- SEE ALSO** `sockLib`, `setsockopt()`

**getw()**

---

**getw()**

|                      |                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>getw()</i> – read the next word (32-bit integer) from a stream                                                                                                                                                                                                                                                                   |
| <b>SYNOPSIS</b>      | <pre>int getw (     FILE * fp /* stream to read from */ )</pre>                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b>   | <p>This routine reads the next 32-bit quantity from a specified stream. It returns EOF on end-of-file or an error; however, this is also a valid integer, thus <i>feof()</i> and <i>ferror()</i> must be used to check for a true end-of-file.</p> <p>This routine is provided for compatibility with earlier VxWorks releases.</p> |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                                                                                                                                                                                                                                      |
| <b>RETURN</b>        | A 32-bit number from the stream, or EOF on either end-of-file or an error.                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>      | <b>ansiStdio, putw()</b>                                                                                                                                                                                                                                                                                                            |

---

**getwd()**

|                    |                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>getwd()</i> – get the current default path                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>char *getwd (     char * pathname /* where to return the pathname */ )</pre>                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>This routine copies the name of the current default path to <i>pathname</i>. It provides the same functionality as <i>ioDefPathGet()</i> and <i>getcwd()</i>. It is provided for compatibility with some older UNIX systems.</p> <p>The parameter <i>pathname</i> should be <b>MAX_FILENAME_LENGTH</b> characters long.</p> |
| <b>RETURNS</b>     | A pointer to the resulting path name.                                                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>ioLib</b>                                                                                                                                                                                                                                                                                                                   |



---

## *gmtime()*

|                      |                                                                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>gmtime()</i> – convert calendar time into UTC broken-down time (ANSI)                                                                                                                                                              |
| <b>SYNOPSIS</b>      | <pre>struct tm *gmtime (     const time_t * timer /* calendar time in seconds */ )</pre>                                                                                                                                              |
| <b>DESCRIPTION</b>   | <p>This routine converts the calendar time pointed to by <i>timer</i> into broken-down time, expressed as Coordinated Universal Time (UTC).</p> <p>This routine is not reentrant. For a reentrant version, see <i>gmtime_r()</i>.</p> |
| <b>INCLUDE FILES</b> | <b>time.h</b>                                                                                                                                                                                                                         |
| <b>RETURNS</b>       | A pointer to a broken-down time structure ( <b>tm</b> ), or a null pointer if UTC is not available.                                                                                                                                   |
| <b>SEE ALSO</b>      | <b>ansiTime</b>                                                                                                                                                                                                                       |

---

## *gmtime\_r()*

|                      |                                                                                                                                                                                                                                                                              |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>gmtime_r()</i> – convert calendar time into broken-down time (POSIX)                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>      | <pre>int gmtime_r (     const time_t * timer,      /* calendar time in seconds */     struct tm *   timeBuffer /* buffer for broken down time */ )</pre>                                                                                                                     |
| <b>DESCRIPTION</b>   | <p>This routine converts the calendar time pointed to by <i>timer</i> into broken-down time, expressed as Coordinated Universal Time (UTC). The broken-down time is stored in <i>timeBuffer</i>.</p> <p>This routine is the POSIX re-entrant version of <i>gmtime()</i>.</p> |
| <b>INCLUDE FILES</b> | <b>time.h</b>                                                                                                                                                                                                                                                                |
| <b>RETURNS</b>       | OK.                                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>      | <b>ansiTime</b>                                                                                                                                                                                                                                                              |

**h()**

---

**h()**

|                    |                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>h()</b> – display or set the size of shell history                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>void h (     int size /* 0 = display, &gt;0 = set history to new size */ )</pre>                                                                                                                                                                          |
| <b>DESCRIPTION</b> | This command displays or sets the size of VxWorks shell history. If no argument is specified, shell history is displayed. If <i>size</i> is specified, that number of the most recent commands is saved for display. The value of <i>size</i> is initially 20. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>shellHistory()</b> , <b>ledLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                   |

---

**hdrBlkBuild()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>hdrBlkBuild()</b> – create the header block and the demuxer information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>void hdrBlkBuild (     SA_HEADER_T * hdr,      /* header block */     VBL_T *      vblist,   /* vblist that was built */     int         opt,      /* reg_option suggesting reg/dereg */     int         group,    /* group ID */     PTR_T       saId      /* ipchandle */ )</pre>                                                                                                                                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>This routine is called to start a process that encodes a message and transmits it to the master agent. Internally, this routine first prepares a header block and demuxer information. These are then passed in to a <b>saMsgBuild()</b> call, along with a varbind list, and a pointer to the IPC mechanism that the master agent can use to respond to this message. As input, <b>hdrBlkBuild()</b> expects:</p> <p><i>hdr</i></p> <p>Expects a pointer to a previously allocated SA_HEADER_T structure. The <b>hdrBlkBuild()</b> routine uses this structure as a storage place within which to build the header block for the message to the master agent.</p> |

*vblist*

Expects a pointer to the **VBL\_T** structure containing the varbind list that you want to include in the message.

*opt*

Expects an operation code that indicates the type of this message. Valid operation codes are as follows:

**SA\_REG\_OBJ\_REQUEST** registers an object with the master agent's MIB tree. The response from the master agent will contain an **SA\_REG\_OBJ\_REPLY** code.

**SA\_REM\_OBJ\_REQUEST** removes (deregisters) an object from the master agent's MIB tree. The response from the master agent will contain an **SA\_REM\_OBJ\_REPLY** code.

**SA\_REG\_INST\_REQUEST** registers an instance with the master agent's MIB tree. The response from the master agent will contain an **SA\_REG\_INST\_REPLY** code.

**SA\_REM\_INST\_REQUEST** removes (deregisters) an instance from the master agent's MIB tree. The response from the master agent will contain an **SA\_REG\_OBJ\_REPLY** code.

**SA\_QUERY\_REQUEST** requests SNMP operations. The response from the master agent will contain an **SA\_QUERY\_REPLY** code.

**SA\_TRAP\_REQUEST** tells the master agent that this message should be handled as a trap. The response from the master agent (if any) will contain an **SA\_TRAP\_REPLY** code.

*group*

Expects the group ID that the master agent has assigned to the objects or instances referenced in *vblist*. This group ID was returned in an **SA\_REG\_OBJ\_REPLY** or an **SA\_REG\_INST\_REPLY** from the master agent. this is an object registration request, you can supply a NULL pointer here.

*said*

Expects a pointer to the IPC mechanism that the master agent can use to respond to the message.

**RETURNS** N/A

**SEE ALSO** **saIoLib**

---

## *help()*

**NAME** *help()* – print a synopsis of selected routines

**SYNOPSIS** `void help (void)`

**help()****DESCRIPTION**

This command prints the following list of the calling sequences for commonly used routines, mostly contained in **usrLib**.

```

help Print this list
dbgHelp Print debug help info
nfsHelp Print nfs help info
netHelp Print network help info
spyHelp Print task histogrammer help info
timexHelp Print execution timer help info
h [n] Print (or set) shell history
i [task] Summary of tasks' TCBS
ti task Complete info on TCB for task
sp adr,args... Spawn a task, pri=100, opt=0, stk=20000
taskSpawn name,pri,opt,stk,adr,args... Spawn a task
td task Delete a task
ts task Suspend a task
tr task Resume a task
d [adr[,nunits[,width]]] Display memory
m adr[,width] Modify memory
mRegs [reg[,task]] Modify a task's registers interactively
pc [task] Return task's program counter
version Print VxWorks version info, and boot line
iam "user"["passwd"] Set user name and passwd
whoami Print user name
cd "path" Set current working path
pwd Print working path
devs List devices
ls ["path"[,long]] List contents of directory
ll ["path"] List contents of directory - long format
rename "old","new" Change name of file
copy ["in"["out"]] Copy in file to out file (0 = std in/out)
ld [syms[,noAbort][,"name"]] Load std in into memory
 (syms = add symbols to table:
 -1 = none, 0 = globals, 1 = all)
lkup ["substr"] List symbols in system symbol table
lkAddr address List symbol table entries near address
checkStack [task] List task stack sizes and usage
printErrno value Print the name of a status value
period secs,adr,args... Spawn task to call function periodically
repeat n,adr,args... Spawn task to call function n times
 (0=forever)
diskFormat "device" Format disk
diskInit "device" Initialize file system on disk
squeeze "device" Squeeze free space on RT-11 device
NOTE: Arguments specifying <task> can be either task ID or name.

```

**RETURNS** N/A**SEE ALSO** *usrLib*, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

2

---

## *hostAdd()*

**NAME** *hostAdd()* – add a host to the host table**SYNOPSIS**

```
STATUS hostAdd
(
 char * hostName, /* host name */
 char * hostAddr /* host addr in standard Internet format */
)
```

**DESCRIPTION** This routine adds a host name to the local host table. This must be called before sockets on the remote host are opened, or before files on the remote host are accessed via **netDrv** or **nfsDrv**.

The host table has one entry per Internet address. More than one name may be used for an address. Additional host names are added as aliases.

**EXAMPLE**

```
-> hostAdd "wrs", "90.2"
-> hostShow
hostname inet address aliases
----- -
localhost 127.0.0.1
yuba 90.0.0.3
wrs 90.0.0.2
value = 12288 = 0x3000 = _bzero + 0x18
```

**RETURNS** OK, or ERROR if the host table is full, the host name/inet address pair is already entered, the Internet address is invalid, or memory is insufficient.**SEE ALSO** *hostLib*, *netDrv*, *nfsDrv*

**hostDelete()**

---

**hostDelete()**

|                    |                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>hostDelete()</i> – delete a host from the host table                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS hostDelete (     char * name, /* host name or alias */     char * addr /* host addr in standard Internet format */ )</pre>                                           |
| <b>DESCRIPTION</b> | This routine deletes a host name from the local host table. If <i>name</i> is a host name, the host entry is deleted. If <i>name</i> is a host name alias, the alias is deleted. |
| <b>RETURNS</b>     | OK, or ERROR if the parameters are invalid or the host is unknown.                                                                                                               |
| <b>SEE ALSO</b>    | <b>hostLib</b>                                                                                                                                                                   |

---

**hostGetByAddr()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>hostGetByAddr()</i> – look up a host in the host table by its Internet address                                                                                                                                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS hostGetByAddr (     int    addr, /* inet address of host */     char * name /* buffer to hold name */ )</pre>                                                                                                                                                                                                                                                                                                                    |
| <b>DESCRIPTION</b> | This routine finds the host name by its Internet address and copies it to <i>name</i> . The buffer <i>name</i> should be preallocated with (MAXHOSTNAMELEN + 1) bytes of memory and is NULL-terminated unless insufficient space is provided. If the DNS resolver library <b>resolvLib</b> has been configured in the vxWorks image, a query for the host name is sent to the DNS server, if the name was not found in the local host table. |
| <b>WARNING</b>     | This routine does not look for aliases. Host names are limited to MAXHOSTNAMELEN (from <b>hostLib.h</b> ) characters.                                                                                                                                                                                                                                                                                                                        |
| <b>RETURNS</b>     | OK, or ERROR if buffer is invalid or the host is unknown.                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <b>hostLib</b> , <i>hostGetByName()</i>                                                                                                                                                                                                                                                                                                                                                                                                      |

---

## *hostGetByName()*

|                    |                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>hostGetByName()</i> – look up a host in the host table by its name                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>int hostGetByName (     char * name /* name of host */ )</pre>                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | This routine returns the Internet address of a host that has been added to the host table by <i>hostAdd()</i> . If the DNS resolver library <b>resolvLib</b> has been configured in the vxWorks image, a query for the host IP address is sent to the DNS server, if the name was not found in the local host table. |
| <b>RETURNS</b>     | The Internet address (as an integer in network byte order), or ERROR if the host is unknown.                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <b>hostLib</b>                                                                                                                                                                                                                                                                                                       |

---

## *hostShow()*

|                    |                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>hostShow()</i> – display the host table                                                   |
| <b>SYNOPSIS</b>    | <pre>void hostShow (void)</pre>                                                              |
| <b>DESCRIPTION</b> | This routine prints a list of remote hosts, along with their Internet addresses and aliases. |
| <b>RETURNS</b>     | N/A                                                                                          |
| <b>SEE ALSO</b>    | <b>netShow</b> , <i>hostAdd()</i>                                                            |

---

## *hostTblInit()*

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <b>NAME</b>     | <i>hostTblInit()</i> – initialize the network host table |
| <b>SYNOPSIS</b> | <pre>void hostTblInit (void)</pre>                       |

***i()***

**DESCRIPTION** This routine initializes the host list data structure used by routines throughout this module. It should be called before any other routines in this module. This is done automatically if the configuration macro `INCLUDE_NET_INIT` is defined.

**RETURNS** N/A

**SEE ALSO** `hostLib`, `usrConfig`

***i()***

**NAME** `i()` – print a summary of each task’s TCB

**SYNOPSIS**

```
void i
(
 int taskNameOrId /* task name or task ID, 0 = summarize all */
)
```

**DESCRIPTION** This command displays a synopsis of all the tasks in the system. The `ti()` routine provides more complete information on a specific task.

Both `i()` and `ti()` use `taskShow()`; see the documentation for `taskShow()` for a description of the output format.

**EXAMPLE**

```
-> i
 NAME ENTRY TID PRI STATUS PC SP ERRNO DELAY

tExcTask _excTask 20fcb00 0 PEND 200c5fc 20fca6c 0 0
tLogTask _logTask 20fb5b8 0 PEND 200c5fc 20fb520 0 0
tShell _shell 20efcac 1 READY 201dc90 20ef980 0 0
tRlogind _rlogind 20f3f90 2 PEND 2038614 20f3db0 0 0
tTelnetd _telnetd 20f2124 2 PEND 2038614 20f2070 0 0
tNetTask _netTask 20f7398 50 PEND 2038614 20f7340 0 0
value = 57 = 0x39 = '9'
```

**CAVEAT** This command should be used only as a debugging aid, since the information is obsolete by the time it is displayed.

**RETURNS** N/A

**SEE ALSO** `usrLib`, `ti()`, `taskShow()`, *VxWorks Programmer’s Guide: Target Shell*, `windsh`, *Tornado User’s Guide: Shell*



---

## ***i0()***

|                    |                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>i0()</i> – return the contents of register <b>i0</b> (also <b>i1</b> – <b>i7</b> ) (SPARC)                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>int i0 (     int taskId /* task ID, 0 means default task */ )</pre>                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b> | <p>This command extracts the contents of in register <b>i0</b> from the TCB of a specified task. If <i>taskId</i> is omitted or 0, the current default task is assumed.</p> <p>Similar routines are provided for all in registers (<b>i0</b> – <b>i7</b>): <i>i0()</i> – <i>i7()</i>.</p> <p>The frame pointer is accessed via <b>i6</b>.</p> |
| <b>RETURNS</b>     | The contents of register <b>i0</b> (or the requested register).                                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <i>dbgArchLib</i> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                           |

---

## ***i8250HrdInit()***

|                    |                                                                                 |
|--------------------|---------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>i8250HrdInit()</i> – initialize the chip                                     |
| <b>SYNOPSIS</b>    | <pre>void i8250HrdInit (     I8250_CHAN * pChan /* pointer to device */ )</pre> |
| <b>DESCRIPTION</b> | This routine is called to reset the chip in a quiescent state.                  |
| <b>RETURNS</b>     | N/A                                                                             |
| <b>SEE ALSO</b>    | <i>i8250Sio</i>                                                                 |

## *i8250Int()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>i8250Int()</i> – handle a receiver/transmitter interrupt                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>void i8250Int (     I8250_CHAN * pChan )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This routine handles four sources of interrupts from the UART. They are prioritized in the following order by the Interrupt Identification Register: Receiver Line Status, Received Data Ready, Transmit Holding Register Empty and Modem Status.</p> <p>If there is another character to be transmitted, it sends it. If not, or if a device has never been created for this channel, just disable the interrupt. disable the interrupt. When a modem status interrupt occurs, the transmit interrupt is enabled if the CTS signal is TRUE.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <i>i8250Sio</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

## *iam()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>iam()</i> – set the remote user name and password                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>STATUS iam (     char * newUser, /* user name to use on remote */     char * newPasswd /* password to use on remote (NULL = none) */ )</pre>                                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b> | <p>This routine specifies the user name that will have access privileges on the remote machine. The user name must exist in the remote machine's <i>/etc/passwd</i>, and if it has been assigned a password, the password must be specified in <i>newPasswd</i>.</p> <p>Either parameter can be NULL, and the corresponding item will not be set.</p> <p>The maximum length of the user name and the password is <b>MAX_IDENTITY_LEN</b> (defined in <b>remLib.h</b>).</p> |

**NOTE** This routine is a more convenient version of *remCurIdSet()* and is intended to be used from the shell.

**RETURNS** OK, or ERROR if the call fails.

**SEE ALSO** *remLib*, *whoami()*, *remCurIdGet()*, *remCurIdSet()*

---

## *icmpShowInit()*

**NAME** *icmpShowInit()* – initialize ICMP show routines

**SYNOPSIS** `void icmpShowInit (void)`

**DESCRIPTION** This routine links the ICMP show facility into the VxWorks system. These routines are included automatically if `INCLUDE_NET_SHOW` and `INCLUDE_ICMP` are defined in `configAll.h`.

**RETURNS** N/A

**SEE ALSO** *icmpShow*

---

## *icmpstatShow()*

**NAME** *icmpstatShow()* – display statistics for ICMP

**SYNOPSIS** `void icmpstatShow (void)`

**DESCRIPTION** This routine displays statistics for the ICMP (Internet Control Message Protocol) protocol.

**RETURNS** N/A

**SEE ALSO** *icmpShow*

## *ideDevCreate()*

- NAME** *ideDevCreate()* – create a device for a IDE disk
- SYNOPSIS**
- ```
BLK_DEV *ideDevCreate
(
    int drive,      /* drive number for hard drive (0 or 1) */
    int nBlocks,   /* device size in blocks (0 = whole disk) */
    int blkOffset /* offset from start of device */
)
```
- DESCRIPTION**
- This routine creates a device for a specified IDE disk.
- drive* is a drive number for the hard drive: it must be 0 or 1.
- The *nBlocks* parameter specifies the size of the device, in blocks. If *nBlocks* is zero, the whole disk is used.
- The *blkOffset* parameter specifies an offset, in blocks, from the start of the device to be used when writing or reading the hard disk. This offset is added to the block numbers passed by the file system during disk accesses. (VxWorks file systems always use block numbers beginning at zero for the start of a device.)
- RETURNS** A pointer to a block device structure (**BLK_DEV**), or NULL if memory cannot be allocated for the device structure.
- SEE ALSO** *ideDrv*, *dosFsMkfs()*, *dosFsDevInit()*, *rt11FsDevInit()*, *rt11FsMkfs()*, *rawFsDevInit()*
-

ideDrv()

- NAME** *ideDrv()* – initialize the IDE driver
- SYNOPSIS**
- ```
STATUS ideDrv
(
 int vector, /* interrupt vector */
 int level, /* interrupt level */
 BOOL manualConfig /* 1 = initialize drive parameters */
)
```
- DESCRIPTION**
- This routine initializes the IDE driver, sets up interrupt vectors, and performs hardware initialization of the IDE chip.

This routine should be called exactly once, before any reads, writes, or calls to *ideDevCreate()*. Normally, it is called by *usrRoot()* in *usrConfig.c*.

The *ideDrv()* call requires a configuration type, *manualConfig*. If this argument is 1, the driver will initialize drive parameters; if the argument is 0, the driver will not initialize drive parameters.

The drive parameters are the number of sectors per track, the number of heads, and the number of cylinders. They are stored in the structure table *ideTypes[]* in *sysLib.c*. The table has two entries: the first is for drive 0; the second is for drive 1. The table has two other members which are used by the driver: the number of bytes per sector and the precompensation cylinder. These two members should be set properly. Definitions of the structure members are:

```
int cylinders; /* number of cylinders */
int heads; /* number of heads */
int sectorsTrack; /* number of sectors per track */
int bytesSector; /* number of bytes per sector */
int precomp; /* precompensation cylinder */
```

**RETURNS** OK, or ERROR if initialization fails.

**SEE ALSO** *ideDrv*, *ideDevCreate()*

---

## *ideRawio()*

**NAME** *ideRawio()* – provide raw I/O access

**SYNOPSIS**

```
STATUS ideRawio
(
 int drive, /* drive number for hard drive (0 or 1) */
 IDE_RAW * pIdeRaw /* pointer to IDE_RAW structure */
)
```

**DESCRIPTION** This routine is called when the raw I/O access is necessary.

*drive* is a drive number for the hard drive: it must be 0 or 1.

The *pIdeRaw* is a pointer to the structure *IDE\_RAW* which is defined in *ideDrv.h*

**RETURNS** OK or ERROR.

**SEE ALSO** *ideDrv*

***ifAddrAdd()***

---

***ifAddrAdd()***

**NAME** *ifAddrAdd()* – Add an interface address for a network interface

**SYNOPSIS**

```
STATUS ifAddrAdd
(
 char * interfaceName, /* name of interface to configure */
 char * interfaceAddress, /* Internet address to assign to interface */
 char * broadcastAddress, /* broadcast address to assign to interface */
 int subnetMask /* subnetMask */
)
```

**DESCRIPTION** This routine assigns an Internet address to a specified network interface. The Internet address can be a host name or a standard Internet address format (e.g., 90.0.0.4). If a host name is specified, it should already have been added to the host table with *hostAdd()*. *interfaceName*, *interfaceAddress* must be specified. *broadcastAddress* is optional. If *broadcastAddress* is NULL, *in\_ifinit()* will generate a *broadcastAddress* by using the *interfaceAddress* and the netmask. *subnetMask* is optional. If *subnetMask* is 0, *in\_ifinit()* will set a *subnetMask* as same as a netmask which is generated by the *interfaceAddress*. *broadcastAddress* is also *destAddress* in case of IFF\_POINTOPOINT.

**RETURNS** OK, or ERROR if the interface cannot be set.

**SEE ALSO** *ifLib*, *ifAddrGet()*, *ifDstAddrSet()*, *ifDstAddrGet()*

---

***ifAddrGet()***

**NAME** *ifAddrGet()* – get the Internet address of a network interface

**SYNOPSIS**

```
STATUS ifAddrGet
(
 char * interfaceName, /* name of interface, i.e. ei0 */
 char * interfaceAddress /* buffer for Internet address */
)
```

**DESCRIPTION** This routine gets the Internet address of a specified network interface and copies it to *interfaceAddress*, which should point to a buffer large enough for INET\_ADDR\_LEN bytes.

**RETURNS** OK or ERROR.

**SEE ALSO** *ifLib*, *ifAddrSet()*, *ifDstAddrSet()*, *ifDstAddrGet()*

---

## *ifAddrSet()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ifAddrSet()</i> – set an interface address for a network interface                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>STATUS ifAddrSet (     char * interfaceName, /* name of interface to configure, i.e. ei0 */     char * interfaceAddress /* Internet address to assign to interface */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This routine assigns an Internet address to a specified network interface. The Internet address can be a host name or a standard Internet address format (e.g., 90.0.0.4). If a host name is specified, it should already have been added to the host table with <i>hostAdd()</i>.</p> <p>A successful call to <i>ifAddrSet()</i> results in the addition of a new route.</p> <p>The subnet mask used in determining the network portion of the address will be that set by <i>ifMaskSet()</i>, or the default class mask if <i>ifMaskSet()</i> has not been called. It is standard practice to call <i>ifMaskSet()</i> prior to calling <i>ifAddrSet()</i>.</p> |
| <b>RETURNS</b>     | OK, or ERROR if the interface cannot be set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SEE ALSO</b>    | <i>ifLib</i> , <i>ifAddrGet()</i> , <i>ifDstAddrSet()</i> , <i>ifDstAddrGet()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

---

## *ifBroadcastGet()*

|                    |                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ifBroadcastGet()</i> – get the broadcast address for a network interface                                                                                     |
| <b>SYNOPSIS</b>    | <pre>STATUS ifBroadcastGet (     char * interfaceName, /* name of interface, i.e. ei0 */     char * broadcastAddress /* buffer for broadcast address */ )</pre> |
| <b>DESCRIPTION</b> | <p>This routine gets the broadcast address for a specified network interface. The broadcast address is copied to the buffer <i>broadcastAddress</i>.</p>        |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                    |
| <b>SEE ALSO</b>    | <i>ifLib</i> , <i>ifBroadcastSet()</i>                                                                                                                          |

---

## **ifBroadcastSet()**

- NAME** *ifBroadcastSet()* – set the broadcast address for a network interface
- SYNOPSIS**
- ```
STATUS ifBroadcastSet
(
    char * interfaceName, /* name of interface to assign, i.e. ei0 */
    char * broadcastAddress /* broadcast address to assign to interface */
)
```
- DESCRIPTION**
- This routine assigns a broadcast address for the specified network interface. The broadcast address must be a string in standard Internet address format (e.g., 90.0.0.0).
- An interface's default broadcast address is its Internet address with a host part of all ones (e.g., 90.255.255.255). This conforms to current ARPA specifications. However, some older systems use an Internet address with a host part of all zeros as a broadcast address.
- NOTE**
- VxWorks automatically accepts a host part of all zeros as a broadcast address, in addition to the default or specified broadcast address. But if VxWorks is to broadcast to older systems using a host part of all zeros as the broadcast address, this routine should be used to change the broadcast address of the interface.
- RETURNS** OK or ERROR.
- SEE ALSO** *ifLib*

ifDstAddrGet()

- NAME** *ifDstAddrGet()* – get the Internet address of a point-to-point peer
- SYNOPSIS**
- ```
STATUS ifDstAddrGet
(
 char * interfaceName, /* name of interface, i.e. ei0 */
 char * dstAddress /* buffer for destination address */
)
```
- DESCRIPTION**
- This routine gets the Internet address of a machine connected to the opposite end of a point-to-point network connection. The Internet address is copied to *dstAddress*.
- RETURNS** OK or ERROR.



SEE ALSO *ifLib, ifDstAddrSet(), ifAddrGet()*

---

## *ifDstAddrSet()*

**NAME** *ifDstAddrSet()* – define an address for the other end of a point-to-point link

**SYNOPSIS**

```
STATUS ifDstAddrSet
(
 char * interfaceName, /* name of interface to configure, i.e. ei0 */
 char * dstAddress /* Internet address to assign to destination */
)
```

**DESCRIPTION** This routine assigns the Internet address of a machine connected to the opposite end of a point-to-point network connection, such as a SLIP connection. Inherently, point-to-point connection-oriented protocols such as SLIP require that addresses for both ends of a connection be specified.

**RETURNS** OK or ERROR.

**SEE ALSO** *ifLib, ifAddrSet(), ifDstAddrGet()*

---

## *ifFlagChange()*

**NAME** *ifFlagChange()* – change the network interface flags

**SYNOPSIS**

```
STATUS ifFlagChange
(
 char * interfaceName, /* name of the network interface, i.e. ei0 */
 int flags, /* the flag to be changed */
 BOOL on /* TRUE=turn on, FALSE=turn off */
)
```

**DESCRIPTION** This routine changes the flags for the specified network interfaces. If the parameter *on* is TRUE, the specified flags are turned on; otherwise, they are turned off. The routines *ifFlagGet()* and *ifFlagSet()* are called to do the actual work.

**RETURNS** OK or ERROR.

**SEE ALSO** *ifLib, ifAddrSet(), ifMaskSet(), ifFlagSet(), ifFlagGet()*

## ***ifFlagGet()***

|                    |                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ifFlagGet()</i> – get the network interface flags                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>STATUS ifFlagGet (     char * interfaceName, /* name of the network interface, i.e. ei0 */     int * flags           /* network flags returned here */ )</pre> |
| <b>DESCRIPTION</b> | This routine gets the flags for a specified network interface. The flags are copied to the buffer <i>flags</i> .                                                    |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                        |
| <b>SEE ALSO</b>    | <i>ifLib</i> , <i>ifFlagSet()</i>                                                                                                                                   |

---

## ***ifFlagSet()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ifFlagSet()</i> – specify the flags for a network interface                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>STATUS ifFlagSet (     char * interfaceName, /* name of the network interface, i.e. ei0 */     int   flags           /* network flags */ )</pre>                                                                                                                                                                                                                                                                                                        |
| <b>DESCRIPTION</b> | <p>This routine changes the flags for a specified network interface. Any combination of the following flags can be specified:</p> <p><b>IFF_UP</b> (0x1)<br/>Brings the network up or down.</p> <p><b>IFF_DEBUG</b> (0x4)<br/>Turns on debugging for the driver interface if supported.</p> <p><b>IFF_LOOPBACK</b> (0x8)<br/>Set for a loopback network.</p> <p><b>IFF_NOTRAILERS</b> (0x20)<br/>Always set (VxWorks does not use the trailer protocol).</p> |

**IFF\_PROMISC** (0x100)

Tells the driver to accept all packets, not just broadcast packets and packets addressed to itself.

**IFF\_ALLMULTI** (0x200)

Tells the driver to accept all multicast packets.

**IFF\_NOARP** (0x80)

Disables ARP for the interface.

**NOTE**

The following flags can only be set at interface initialization time. Specifying these flags does not change any settings in the interface data structure.

**IFF\_POINTOPOINT** (0x10)

Identifies a point-to-point interface such as PPP or SLIP.

**IFF\_RUNNING** (0x40)

Set when the device turns on.

**IFF\_BROADCAST** (0x2)

Identifies a broadcast interface.

**RETURNS**

OK or ERROR.

**SEE ALSO**

*ifLib*, *ifFlagChange()*, *ifFlagGet()*

---

## *ifMaskGet()*

**NAME**

*ifMaskGet()* – get the subnet mask for a network interface

**SYNOPSIS**

```
STATUS ifMaskGet
(
 char * interfaceName, /* name of interface, i.e. ei0 */
 int * netMask /* buffer for subnet mask */
)
```

**DESCRIPTION**

This routine gets the subnet mask for a specified network interface. The subnet mask is copied to the buffer *netMask*. The subnet mask is returned in host byte order.

**RETURNS**

OK or ERROR.

**SEE ALSO**

*ifLib*, *ifAddrGet()*, *ifFlagGet()*

***ifMaskSet()***

---

***ifMaskSet()*****NAME** *ifMaskSet()* – define a subnet for a network interface**SYNOPSIS**  

```
STATUS ifMaskSet
(
 char * interfaceName, /* name of interface to set mask for, i.e. ei0 */
 int netMask /* subnet mask (e.g. 0xff000000) */
)
```

**DESCRIPTION**  
This routine allocates additional bits to the network portion of an Internet address. The network portion is specified with a mask that must contain ones in all positions that are to be interpreted as the network portion. This includes all the bits that are normally interpreted as the network portion for the given class of address, plus the bits to be added. Note that all bits must be contiguous. The mask is specified in host byte order.

In order to correctly interpret the address, a subnet mask should be set for an interface prior to setting the Internet address of the interface with the routine *ifAddrSet()*.

**RETURNS** OK or ERROR.**SEE ALSO** *ifLib*, *ifAddrSet()*

---

***ifMetricGet()*****NAME** *ifMetricGet()* – get the metric for a network interface**SYNOPSIS**  

```
STATUS ifMetricGet
(
 char * interfaceName, /* name of the network interface, i.e. ei0 */
 int * pMetric /* returned interface's metric */
)
```

**DESCRIPTION**  
This routine retrieves the metric for a specified network interface. The metric is copied to the buffer *pMetric*.**RETURNS** OK or ERROR.**SEE ALSO** *ifLib*, *ifMetricSet()*

---

## *ifMetricSet()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ifMetricSet()</i> – specify a network interface hop count                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>STATUS ifMetricSet (     char * interfaceName, /* name of the network interface, i.e. ei0 */     int    metric         /* metric for this interface */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>DESCRIPTION</b> | This routine configures <i>metric</i> for a network interface from the host machine to the destination network. This information is used primarily by the IP routing algorithm to compute the relative distance for a collection of hosts connected to each interface. For example, a higher <i>metric</i> for SLIP interfaces can be specified to discourage routing a packet to slower serial line connections. Note that when <i>metric</i> is zero, the IP routing algorithm allows for the direct sending of a packet having an IP network address that is not necessarily the same as the local network address. |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>ifLib</i> , <i>ifMetricGet()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

---

## *ifRouteDelete()*

|                    |                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ifRouteDelete()</i> – delete routes associated with a network interface                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>int ifRouteDelete (     char * ifName, /* name of the interface */     int    unit    /* unit number for this interface */ )</pre>                                                                                                                                                                     |
| <b>DESCRIPTION</b> | This routine deletes all routes that have been associated with the specified interface. A route is associated with an interface if its destination equals to the assigned address, or network number. This routine does not remove routes to arbitrary destinations which pass through the given interface. |
| <b>RETURNS</b>     | The number of routes deleted, or ERROR if an interface is not specified.                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <i>ifLib</i>                                                                                                                                                                                                                                                                                                |

---

## **ifShow()**

**NAME** *ifShow()* – display the attached network interfaces

**SYNOPSIS**

```
void ifShow
(
 char * ifName /* name of the interface to show */
)
```

**DESCRIPTION** This routine displays the attached network interfaces for debugging and diagnostic purposes. If *ifName* is given, only the interfaces belonging to that group are displayed. If *ifName* is omitted, all attached interfaces are displayed.

For each interface selected, the following are shown: Internet address, point-to-point peer address (if using SLIP), broadcast address, netmask, subnet mask, Ethernet address, route metric, maximum transfer unit, number of packets sent and received on this interface, number of input and output errors, and flags (such as loopback, point-to-point, broadcast, promiscuous, ARP, running, and debug).

**EXAMPLE** The following call displays all interfaces whose names begin with "ln", (such as "ln0", "ln1", and "ln2"):

```
-> ifShow "ln"
```

The following call displays just the interface "ln0":

```
-> ifShow "ln0"
```

**RETURNS** N/A

**SEE ALSO** *netShow*, *routeShow()*, *ifLib*

---

## **ifunit()**

**NAME** *ifunit()* – map an interface name to an interface structure pointer

**SYNOPSIS**

```
struct ifnet *ifunit
(
 char * ifname /* name of the interface */
)
```

**DESCRIPTION** This routine returns a pointer to a network interface structure for *name* or NULL if no such interface exists. For example:

```
struct ifnet *pIf;
...
pIf = ifunit ("ln0");
```

*pIf* points to the data structure that describes the first network interface device if *ln0* is mapped successfully.

**RETURNS** A pointer to the interface structure, or NULL if an interface is not found.

**SEE ALSO** *ifLib*, *etherLib*

---

## *igmpShowInit()*

**NAME** *igmpShowInit()* – initialize IGMP show routines

**SYNOPSIS** `void igmpShowInit (void)`

**DESCRIPTION** This routine links the IGMP show facility into the VxWorks system. These routines are included automatically if `INCLUDE_NET_SHOW` and `INCLUDE_IGMP` are defined in `configAll.h`.

**RETURNS** N/A

**SEE ALSO** *igmpShow*

---

## *igmpstatShow()*

**NAME** *igmpstatShow()* – display statistics for IGMP

**SYNOPSIS** `void igmpstatShow (void)`

**DESCRIPTION** This routine displays statistics for the IGMP (Internet Group Management Protocol) protocol.

**RETURNS** N/A

**SEE ALSO** *igmpShow*

## *index()*

**NAME** *index()* – find the first occurrence of a character in a string

**SYNOPSIS**

```
char *index
(
 const char * s, /* string in which to find character */
 int c /* character to find in string */
)
```

**DESCRIPTION** This routine finds the first occurrence of character *c* in string *s*.

**RETURNS** A pointer to the located character, or NULL if *c* is not found.

**SEE ALSO** *bLib*, *strchr()*.

---

## *inet\_addr()*

**NAME** *inet\_addr()* – convert a dot notation Internet address to a long integer

**SYNOPSIS**

```
u_long inet_addr
(
 char * inetString /* string inet address */
)
```

**DESCRIPTION** This routine interprets an Internet address. All the network library routines call this routine to interpret entries in the data bases which are expected to be an address. The value returned is in network order.

**EXAMPLE** The following example returns 0x5a000002:

```
inet_addr ("90.0.0.2");
```

**RETURNS** The Internet address, or ERROR.

**SEE ALSO** *inetLib*



---

## *inet\_aton()*

|                    |                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>inet_aton()</i> – convert a network address from dot notation, store in a structure                                                                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>STATUS inet_aton (     char *          pString, /* string containing address, dot notation */     struct in_addr * inetAddress /* struct in which to store address */ )</pre>                                                                       |
| <b>DESCRIPTION</b> | This routine interprets an Internet address. All the network library routines call this routine to interpret entries in the data bases that are expected to be an address. The value returned is stored in network byte order in the structure provided. |
| <b>EXAMPLE</b>     | The following example returns 0x5a000002 in the <i>s_addr</i> member of the structure pointed to by <i>pinetAddr</i> :<br><pre>inet_addr ("90.0.0.2", pinetAddr);</pre>                                                                                  |
| <b>RETURNS</b>     | OK, or ERROR.                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <i>inetLib</i>                                                                                                                                                                                                                                           |

---

## *inet\_lnaof()*

|                    |                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>inet_lnaof()</i> – get the local address (host number) from the Internet address                                                          |
| <b>SYNOPSIS</b>    | <pre>int inet_lnaof (     int inetAddress /* inet addr from which to extract local portion */ )</pre>                                        |
| <b>DESCRIPTION</b> | This routine returns the local network address portion of an Internet address. The routine handles class A, B, and C network number formats. |
| <b>EXAMPLE</b>     | The following example returns 2:<br><pre>inet_lnaof (0x5a000002);</pre>                                                                      |
| <b>RETURNS</b>     | The local address portion of <i>inetAddress</i> .                                                                                            |
| <b>SEE ALSO</b>    | <i>inetLib</i>                                                                                                                               |

## *inet\_makeaddr()*

- NAME** *inet\_makeaddr()* – form an Internet address from network and host numbers
- SYNOPSIS**
- ```
struct in_addr inet_makeaddr
(
    int netAddr, /* network part of the address */
    int hostAddr /* host part of the address */
)
```
- DESCRIPTION** This routine constructs the Internet address from the network number and local host address.
- WARNING** This routine is supplied for UNIX compatibility only. Each time this routine is called, four bytes are allocated from memory. Use *inet_makeaddr_b()* instead.
- EXAMPLE** The following example returns the address 0x5a000002 to the structure **in_addr**:
- ```
inet_makeaddr (0x5a, 2);
```
- RETURNS** The network address in an **in\_addr** structure.
- SEE ALSO** *inetLib*, *inet\_makeaddr\_b()*

---

## *inet\_makeaddr\_b()*

- NAME** *inet\_makeaddr\_b()* – form an Internet address from network and host numbers
- SYNOPSIS**
- ```
void inet_makeaddr_b
(
    int          netAddr, /* network part of the inet address */
    int          hostAddr, /* host part of the inet address */
    struct in_addr * pInetAddr /* where to return the inet address */
)
```
- DESCRIPTION** This routine constructs the Internet address from the network number and local host address. This routine is identical to the UNIX *inet_makeaddr()* routine except that you must provide a buffer for the resulting value.
- EXAMPLE** The following copies the address 0x5a000002 to the location pointed to by *pInetAddr*:

```
inet_makeaddr_b (0x5a, 2, pInetAddress);
```

RETURNS N/A

SEE ALSO `inetLib`

inet_netof()

NAME *inet_netof()* – return the network number from an Internet address

SYNOPSIS

```
int inet_netof
(
    struct in_addr inetAddress /* inet address */
)
```

DESCRIPTION This routine extracts the network portion of an Internet address.

EXAMPLE The following example returns 0x5a:

```
inet_netof (0x5a000002);
```

RETURNS The network portion of *inetAddress*.

SEE ALSO `inetLib`

inet_netof_string()

NAME *inet_netof_string()* – extract the network address in dot notation

SYNOPSIS

```
void inet_netof_string
(
    char * inetString, /* inet addr to extract local portion from */
    char * netString /* net inet address to return */
)
```

DESCRIPTION This routine extracts the network Internet address from a host Internet address (specified in dotted decimal notation). The routine handles class A, B, and C network addresses. The buffer *netString* should be `INET_ADDR_LEN` bytes long.

inet_network()

- NOTE** This is the only routine in **inetLib** that handles subnet masks correctly.
- EXAMPLE** The following example copies "90.0.0.0" to *netString*:
- ```
inet_netof_string ("90.0.0.2", netString);
```
- RETURNS** N/A
- SEE ALSO** **inetLib**

---

***inet\_network()***

- NAME** *inet\_network()* – convert an Internet network number from string to address
- SYNOPSIS**
- ```
u_long inet_network  
(  
    char * inetString /* string version of inet addr */  
)
```
- DESCRIPTION** This routine forms a network address from an ASCII string containing an Internet network number.
- EXAMPLE** The following example returns 0x5a:
- ```
inet_network ("90");
```
- RETURNS** The Internet address version of an ASCII string.
- SEE ALSO** **inetLib**

---

***inet\_ntoa()***

- NAME** *inet\_ntoa()* – convert a network address to dotted decimal notation
- SYNOPSIS**
- ```
char *inet_ntoa  
(  
    struct in_addr inetAddress /* inet address */  
)
```
- DESCRIPTION** This routine converts an Internet address in network format to dotted decimal notation.

WARNING This routine is supplied for UNIX compatibility only. Each time this routine is called, 18 bytes are allocated from memory. Use *inet_ntoa_b()* instead.

EXAMPLE The following example returns a pointer to the string "90.0.0.2":

```
struct in_addr iaddr;  
...  
iaddr.s_addr = 0x5a000002;  
...  
inet_ntoa (iaddr);
```

RETURNS A pointer to the string version of an Internet address.

SEE ALSO *inetLib*, *inet_ntoa_b()*

inet_ntoa_b()

NAME *inet_ntoa_b()* – convert an network address to dot notation, store it in a buffer

SYNOPSIS

```
void inet_ntoa_b  
(  
    struct in_addr inetAddress, /* inet address */  
    char *          pString     /* where to return ASCII string */  
)
```

DESCRIPTION This routine converts an Internet address in network format to dotted decimal notation. This routine is identical to the UNIX *inet_ntoa()* routine except that you must provide a buffer of size `INET_ADDR_LEN`.

EXAMPLE The following example copies the string "90.0.0.2" to *pString*:

```
struct in_addr iaddr;  
...  
iaddr.s_addr = 0x5a000002;  
...  
inet_ntoa_b (iaddr, pString);
```

RETURNS N/A

SEE ALSO *inetLib*

inetstatShow()

| | |
|--------------------|--|
| NAME | <i>inetstatShow()</i> – display all active connections for Internet protocol sockets |
| SYNOPSIS | <code>void inetstatShow (void)</code> |
| DESCRIPTION | This routine displays a list of all active Internet protocol sockets in a format similar to the UNIX netstat command. |
| RETURNS | N/A |
| SEE ALSO | netShow |

infinity()

| | |
|----------------------|---|
| NAME | <i>infinity()</i> – return a very large double |
| SYNOPSIS | <code>double infinity (void)</code> |
| DESCRIPTION | This routine returns a very large double. |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision representation of positive infinity. |
| SEE ALSO | mathALib |

infinityf()

| | |
|----------------------|--|
| NAME | <i>infinityf()</i> – return a very large float |
| SYNOPSIS | <code>float infinityf (void)</code> |
| DESCRIPTION | This routine returns a very large float. |
| INCLUDE FILES | math.h |

RETURNS The single-precision representation of positive infinity.

SEE ALSO `mathALib`

inflate()

NAME *inflate()* – inflate compressed code

SYNOPSIS

```
int inflate
(
    Byte * src,
    Byte * dest,
    int   nBytes
)
```

DESCRIPTION This routine inflates *nBytes* of data starting at address *src*. The inflated code is copied starting at address *dest*. Two sanity checks are performed on the data being decompressed. First, we look for a magic number at the start of the data to verify that it is really a compressed stream. Second, the entire data is optionally checksummed to verify its integrity. By default, the checksum is not verified in order to speed up the booting process. To turn on checksum verification, set the global variable `inflateCksum` to TRUE in the BSP.

RETURNS OK or ERROR.

SEE ALSO `inflateLib`

intConnect()

NAME *intConnect()* – connect a C routine to a hardware interrupt

SYNOPSIS

```
STATUS intConnect
(
    VOIDFUNCPTR * vector, /* interrupt vector to attach to */
    VOIDFUNCPTR  routine, /* routine to be called */
    int          parameter /* parameter to be passed to routine */
)
```

intContext()

| | |
|--------------------|---|
| DESCRIPTION | <p>This routine connects a specified C routine to a specified interrupt vector. The address of <i>routine</i> is generally stored at <i>vector</i> so that <i>routine</i> is called with <i>parameter</i> when the interrupt occurs. The routine is invoked in supervisor mode at interrupt level. A proper C environment is established, the necessary registers saved, and the stack set up.</p> <p>The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.</p> <p>This routine generally simply calls <i>intHandlerCreate()</i> and <i>intVecSet()</i>. The address of the handler returned by <i>intHandlerCreate()</i> is what actually goes in the interrupt vector.</p> <p>This routine takes an interrupt vector as a parameter, which is the byte offset into the vector table. Macros are provided to convert between interrupt vectors and interrupt numbers, see intArchLib.</p> |
| NOTE ARM | ARM processors generally do not have on-chip interrupt controllers. Control of interrupts is a BSP-specific matter. This routine calls a BSP-specific routine to install the handler such that, when the interrupt occurs, <i>routine</i> is called with <i>parameter</i> . |
| RETURNS | OK, or ERROR if the interrupt handler cannot be built. |
| SEE ALSO | intArchLib , <i>intHandlerCreate()</i> , <i>intVecSet()</i> |

intContext()

| | |
|--------------------|---|
| NAME | <i>intContext()</i> – determine if the current state is in interrupt or task context |
| SYNOPSIS | <code>BOOL intContext (void)</code> |
| DESCRIPTION | This routine returns TRUE only if the current execution state is in interrupt context and not in a meaningful task context. |
| RETURNS | TRUE or FALSE. |
| SEE ALSO | intLib |

intCount()

| | |
|--------------------|--|
| NAME | <i>intCount()</i> – get the current interrupt nesting depth |
| SYNOPSIS | <code>int intCount (void)</code> |
| DESCRIPTION | This routine returns the number of interrupts that are currently nested. |
| RETURNS | The number of nested interrupts. |
| SEE ALSO | <code>intLib</code> |

intCRGet()

| | |
|--------------------|---|
| NAME | <i>intCRGet()</i> – read the contents of the cause register (MIPS) |
| SYNOPSIS | <code>int intCRGet (void)</code> |
| DESCRIPTION | This routine reads and returns the contents of the MIPS cause register. |
| RETURNS | The contents of the cause register. |
| SEE ALSO | <code>intArchLib</code> |

intCRSet()

| | |
|--------------------|---|
| NAME | <i>intCRSet()</i> – write the contents of the cause register (MIPS) |
| SYNOPSIS | <pre>void intCRSet (int value /* value to write to cause register */)</pre> |
| DESCRIPTION | This routine writes the contents of the MIPS cause register. |
| RETURNS | N/A |
| SEE ALSO | <code>intArchLib</code> |

intDisable()

| | |
|--------------------|--|
| NAME | <i>intDisable()</i> – disable corresponding interrupt bits (MIPS, PowerPC, ARM) |
| SYNOPSIS | <pre>int intDisable (int level /* new interrupt bits (0x0 - 0xff00) */)</pre> |
| DESCRIPTION | On MIPS and PowerPC architectures, this routine disables the corresponding interrupt bits from the present status register. |
| NOTE ARM | ARM processors generally do not have on-chip interrupt controllers. Control of interrupts is a BSP-specific matter. This routine calls a BSP-specific routine to disable a particular interrupt level, regardless of the current interrupt mask level. |
| NOTE MIPS | For MIPS, the macros <code>SR_IBIT1</code> – <code>SR_IBIT8</code> define bits that may be set. |
| RETURNS | OK or ERROR. (MIPS: The previous contents of the status register). |
| SEE ALSO | <code>intArchLib</code> |

intEnable()

| | |
|--------------------|--|
| NAME | <i>intEnable()</i> – enable corresponding interrupt bits (MIPS, PowerPC, ARM) |
| SYNOPSIS | <pre>int intEnable (int level /* new interrupt bits (0x00 - 0xff00) */)</pre> |
| DESCRIPTION | This routine enables the input interrupt bits on the present status register of the MIPS and PowerPC processors. |
| NOTE ARM | ARM processors generally do not have on-chip interrupt controllers. Control of interrupts is a BSP-specific matter. This routine calls a BSP-specific routine to enable the interrupt. For each interrupt level to be used, there must be a call to this routine before it will be allowed to interrupt. |
| NOTE MIPS | For MIPS, it is strongly advised that the level be a combination of <code>SR_IBIT1</code> – <code>SR_IBIT8</code> . |

RETURNS OK or ERROR. (MIPS: The previous contents of the status register).

SEE ALSO `intArchLib`

intHandlerCreate()

NAME *intHandlerCreate()* – construct an interrupt handler for a C routine (MC680x0, SPARC, i960, x86, MIPS)

SYNOPSIS

```
FUNCPTR intHandlerCreate
(
    FUNCPTR routine, /* routine to be called */
    int     parameter /* parameter to be passed to routine */
)
```

DESCRIPTION This routine builds an interrupt handler around the specified C routine. This interrupt handler is then suitable for connecting to a specific vector address with *intVecSet()*. The interrupt handler is invoked in supervisor mode at interrupt level. A proper C environment is established, the necessary registers saved, and the stack set up.

The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.

RETURNS A pointer to the new interrupt handler, or NULL if memory is insufficient.

SEE ALSO `intArchLib`

intLevelSet()

NAME *intLevelSet()* – set the interrupt level (MC680x0, SPARC, i960, x86, ARM)

SYNOPSIS

```
int intLevelSet
(
    int level /* new interrupt level mask */
)
```

intLock()

| | |
|--------------------|---|
| DESCRIPTION | This routine changes the interrupt mask in the status register to take on the value specified by <i>level</i> . Interrupts are locked out at or below that level. The value of <i>level</i> must be in the following range: MC680x0: 0 – 7 SPARC: 0 – 15 i960: 0 – 31 ARM BSP-specific On SPARC systems, traps must be enabled before the call. |
| WARNING | Do not call VxWorks system routines with interrupts locked. Violating this rule may re-enable interrupts unpredictably. |
| RETURNS | The previous interrupt level. |
| SEE ALSO | intArchLib |

intLock()

| | |
|-----------------------|---|
| NAME | <i>intLock()</i> – lock out interrupts |
| SYNOPSIS | <code>int intLock (void)</code> |
| DESCRIPTION | <p>This routine disables interrupts. The <i>intLock()</i> routine returns an architecture-dependent lock-out key representing the interrupt level prior to the call; this key can be passed to <i>intUnlock()</i> to re-enable interrupts.</p> <p>For MC680x0, SPARC, i960, and i386/i486 architectures, interrupts are disabled at the level set by <i>intLockLevelSet()</i>. The default lock-out level is the highest interrupt level (MC680x0 = 7, SPARC = 15, i960 = 31, i386/i486 = 1).</p> <p>For MIPS processors, interrupts are disabled at the master lock-out level; this means no interrupt can occur even if unmasked in the IntMask bits (15-8) of the status register.</p> <p>For ARM processors, interrupts (IRQs) are disabled by setting the I bit in the CPSR. This means no IRQs can occur.</p> <p>For PowerPC processors, there is only one interrupt vector. The external interrupt (vector offset 0x500) is disabled when <i>intLock()</i> is called; this means that the processor cannot be interrupted by any external event.</p> |
| IMPLEMENTATION | <p>The lock-out key is implemented differently for different architectures:</p> <p> MC680x0: interrupt field mask</p> |

SPARC: interrupt level (0 – 15)
i960: interrupt level (0 – 31)
MIPS: status register
i386/i486: interrupt enable flag (IF) bit from EFLAGS register
PowerPC: MSR register value
ARM I bit from the CPSR

WARNINGS

Do not call VxWorks system routines with interrupts locked. Violating this rule may re-enable interrupts unpredictably.

The routine *intLock()* can be called from either interrupt or task level. When called from a task context, the interrupt lock level is part of the task context. Locking out interrupts does not prevent rescheduling. Thus, if a task locks out interrupts and invokes kernel services that cause the task to block (e.g., *taskSuspend()* or *taskDelay()*) or that cause a higher priority task to be ready (e.g., *semGive()* or *taskResume()*), then rescheduling occurs and interrupts are unlocked while other tasks run. Rescheduling may be explicitly disabled with *taskLock()*. Traps must be enabled when calling this routine.

EXAMPLES

```
lockKey = intLock ();  
... (work with interrupts locked out)  
intUnlock (lockKey);
```

To lock out interrupts and task scheduling as well (see WARNING above):

```
if (taskLock() == OK)  
{  
    lockKey = intLock ();  
    ... (critical section)  
    intUnlock (lockKey);  
    taskUnlock();  
}  
else  
{  
    ... (error message or recovery attempt)  
}
```

RETURNS

An architecture-dependent lock-out key for the interrupt level prior to the call.

SEE ALSO

intArchLib, *intUnlock()*, *taskLock()*, *intLockLevelSet()*

intLockLevelGet()

| | |
|--------------------|---|
| NAME | <i>intLockLevelGet()</i> – get the current interrupt lock-out level (MC680x0, SPARC, i960, x86, ARM) |
| SYNOPSIS | <pre>int intLockLevelGet (void)</pre> |
| DESCRIPTION | This routine returns the current interrupt lock-out level, which is set by <i>intLockLevelSet()</i> and stored in the globally accessible variable intLockMask . This is the interrupt level currently masked when interrupts are locked out by <i>intLock()</i> . The default lock-out level (MC680x0 = 7, SPARC = 15, i960 = 31, i386/i486 = 1) is initially set by <i>kernelInit()</i> when VxWorks is initialized. |
| RETURNS | The interrupt level currently stored in the interrupt lock-out mask. (ARM = ERROR always) |
| SEE ALSO | <i>intArchLib</i> , <i>intLockLevelSet()</i> |

intLockLevelSet()

| | |
|--------------------|--|
| NAME | <i>intLockLevelSet()</i> – set the current interrupt lock-out level (MC680x0, SPARC, i960, x86, ARM) |
| SYNOPSIS | <pre>void intLockLevelSet (int newLevel /* new interrupt level */)</pre> |
| DESCRIPTION | This routine sets the current interrupt lock-out level and stores it in the globally accessible variable intLockMask . The specified interrupt level is masked when interrupts are locked by <i>intLock()</i> . The default lock-out level (MC680x0 = 7, SPARC = 15, i960 = 31, i386/i486 = 1) is initially set by <i>kernelInit()</i> when VxWorks is initialized. |
| NOTE ARM | On the ARM, this call establishes the interrupt level to be set when <i>intLock()</i> is called. |
| RETURNS | N/A |
| SEE ALSO | <i>intArchLib</i> , <i>intLockLevelGet()</i> , <i>intLock()</i> , <i>taskLock()</i> |

intSRGet()

NAME *intSRGet()* – read the contents of the status register (MIPS)

SYNOPSIS `int intSRGet (void)`

DESCRIPTION This routine reads and returns the contents of the MIPS status register.

RETURNS The previous contents of the status register.

SEE ALSO `intArchLib`

intSRSet()

NAME *intSRSet()* – update the contents of the status register (MIPS)

SYNOPSIS `int intSRSet
(
 int value /* value to write to status register */
)`

DESCRIPTION This routine updates and returns the previous contents of the MIPS status register.

RETURNS The previous contents of the status register.

SEE ALSO `intArchLib`

intUninitVecSet()

NAME *intUninitVecSet()* – set the uninitialized vector handler (ARM)

SYNOPSIS `void intUninitVecSet
(
 VOIDFUNCPTR routine /* ptr to user routine */
)`

DESCRIPTION This routine installs a handler for the uninitialized vectors to be called when any uninitialised vector is entered.

RETURNS N/A.

SEE ALSO **intArchLib**

intUnlock()

NAME *intUnlock()* – cancel interrupt locks

SYNOPSIS

```
void intUnlock
(
    int lockKey /* lock-out key returned by preceding intLock() */
)
```

DESCRIPTION This routine re-enables interrupts that have been disabled by *intLock()*. The parameter *lockKey* is an architecture-dependent lock-out key returned by a preceding *intLock()* call.

RETURNS N/A

SEE ALSO **intArchLib**, **intLock()**

intVecBaseGet()

NAME *intVecBaseGet()* – get the vector (trap) base address (MC680x0, SPARC, i960, x86, MIPS, ARM)

SYNOPSIS **FUNCPTR *intVecBaseGet (void)**

DESCRIPTION This routine returns the current vector base address, which is set with *intVecBaseSet()*.

RETURNS The current vector base address (i960 = value of **sysIntTableset** in **sysLib**, MIPS = 0 always, ARM = 0 always).

SEE ALSO **intArchLib**, **intVecBaseSet()**

intVecBaseSet()

| | |
|--------------------|--|
| NAME | <i>intVecBaseSet()</i> – set the vector (trap) base address (MC680x0, SPARC, i960, x86, MIPS, ARM) |
| SYNOPSIS | <pre>void intVecBaseSet (FUNCPTR * baseAddr /* new vector (trap) base address */)</pre> |
| DESCRIPTION | This routine sets the vector (trap) base address. The CPU's vector base register is set to the specified value, and subsequent calls to <i>intVecGet()</i> or <i>intVecSet()</i> will use this base address. The vector base address is initially 0 (0x1000 for SPARC), until modified by calls to this routine. |
| NOTE SPARC | On SPARC processors, the vector base address must be on a 4 Kbyte boundary (that is, its bottom 12 bits must be zero). |
| NOTE 68000 | The 68000 has no vector base register; thus, this routine is a no-op for 68000 systems. |
| NOTE I960 | This routine is a no-op for i960 systems. The interrupt vector table is located in sysLib , and moving it by <i>intVecBaseSet()</i> would require resetting the processor. Also, the vector base is cached on-chip in the PRCB and thus cannot be set from this routine. |
| NOTE MIPS | The MIPS processors have no vector base register; thus this routine is a no-op for this architecture. |
| NOTE ARM | The ARM processors have no vector base register; thus this routine is a no-op for this architecture. |
| RETURNS | N/A |
| SEE ALSO | <i>intArchLib</i> , <i>intVecBaseGet()</i> , <i>intVecGet()</i> , <i>intVecSet()</i> |

intVecGet()

intVecGet()

- NAME** *intVecGet()* – get an interrupt vector (MC680x0, SPARC, i960, x86, MIPS)
- SYNOPSIS**
- ```

FUNCPTR intVecGet
(
 FUNCPTR * vector /* vector offset */
)

```
- DESCRIPTION**
- This routine returns a pointer to the exception/interrupt handler attached to a specified vector. The vector is specified as an offset into the CPU's vector table. This vector table starts, by default, at:
- ```

MC680x0:  0
SPARC:    0x1000
i960:     sysIntTable in sysLib
MIPS:     excBsrTbl in excArchLib
i386/i486: 0

```
- However, the vector table may be set to start at any address with *intVecBaseSet()* (on CPUs for which it is available).
- This routine takes an interrupt vector as a parameter, which is the byte offset into the vector table. Macros are provided to convert between interrupt vectors and interrupt numbers, see **intArchLib**.
- NOTE 1960** The interrupt table location is reinitialized to *sysIntTable* after booting. This location is returned by *intVecBaseGet()*.
- RETURNS** A pointer to the exception/interrupt handler attached to the specified vector.
- SEE ALSO** **intArchLib**, *intVecSet()*, *intVecBaseSet()*

intVecSet()

- NAME** *intVecSet()* – set a CPU vector (trap) (MC680x0, SPARC, i960, x86, MIPS)
- SYNOPSIS**
- ```

void intVecSet
(
 FUNCPTR * vector, /* vector offset */
 FUNCPTR function /* address to place in vector */
)

```

**DESCRIPTION** This routine attaches an exception/interrupt/trap handler to a specified vector. The vector is specified as an offset into the CPU's vector table. This vector table starts, by default, at:

```
MC680x0: 0
SPARC: 0x1000
i960: sysIntTable in sysLib
MIPS: excBsrTbl in excArchLib
i386/i486: 0
```

However, the vector table may be set to start at any address with *intVecBaseSet()* (on CPUs for which it is available). The vector table is set up in *usrInit()*.

This routine takes an interrupt vector as a parameter, which is the byte offset into the vector table. Macros are provided to convert between interrupt vectors and interrupt numbers, see **intArchLib**.

**NOTE SPARC** This routine generates code to:

- (1) save volatile registers;
- (2) fix possible window overflow;
- (3) read the processor state register into register %L0; and
- (4) jump to the specified address.

The *intVecSet()* routine puts this generated code into the trap table entry corresponding to *vector*.

Window overflow and window underflow are sacred to the kernel and may not be pre-empted. They are written here only to track changing trap base registers (TBRs). With the "branch anywhere" scheme (as opposed to the branch PC-relative +/-8 megabytes) the first instruction in the vector table must not be a change of flow control nor affect any critical registers. The JMPL that replaces the BA will always execute the next vector's first instruction.

**NOTE I960** Vectors 0-7 are illegal vectors; using them puts the vector into the priorities/pending portion of the table, which yields undesirable actions. The i960CA caches the NMI vector in internal RAM at system power-up. This is where the vector is taken when the NMI occurs. Thus, it is important to check to see if the vector being changed is the NMI vector, and, if so, to write it to internal RAM.

**NOTE MIPS** On MIPS CPUs the vector table is set up statically in software.

**RETURNS** N/A

**SEE ALSO** **intArchLib**, *intVecBaseSet()*, *intVecGet()*

---

## *intVecTableWriteProtect()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>intVecTableWriteProtect()</i> – write-protect exception vector table (MC680x0, SPARC, i960, x86, ARM)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <b>STATUS</b> <code>intVecTableWriteProtect (void)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>If the unbundled Memory Management Unit (MMU) support package (VxVMI) is present, this routine write-protects the exception vector table to protect it from being accidentally corrupted.</p> <p>Note that other data structures contained in the page will also be write-protected. In the default VxWorks configuration, the exception vector table is located at location 0 in memory. Write-protecting this affects the backplane anchor, boot configuration information, and potentially the text segment (assuming the default text location of 0x1000.) All code that manipulates these structures has been modified to write-enable memory for the duration of the operation. If you select a different address for the exception vector table, be sure it resides in a page separate from other writable data structures.</p> |
| <b>RETURNS</b>     | OK, or ERROR if memory cannot be write-protected.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>ERRNO</b>       | S_intLib_VEC_TABLE_WP_UNAVAILABLE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <code>intArchLib</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

---

## *ioctl()*

|                    |                                                                                                                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ioctl()</i> – perform an I/O control function                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>int ioctl (     int fd,          /* file descriptor */     int function,   /* function code */     int arg          /* arbitrary argument */ )</pre>                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This routine performs an I/O control function on a device. The control functions used by VxWorks device drivers are defined in the header file <code>ioLib.h</code>. Most requests are passed on to the driver for handling. Since the availability of <i>ioctl()</i> functions is driver-specific,</p> |

these functions are discussed separately in **tyLib**, **pipeDrv**, **nfsDrv**, **dosFsLib**, **rt11FsLib**, and **rawFsLib**.

The following example renames the file or directory to the string "newname":

```
ioctl (fd, FIORENAME, "newname");
```

Note that the function **FIOGETNAME** is handled by the I/O interface level and is not passed on to the device driver itself. Thus this function code value should not be used by customer-written drivers.

**RETURNS** The return value of the driver, or **ERROR** if the file descriptor does not exist.

**SEE ALSO** **ioLib**, **tyLib**, **pipeDrv**, **nfsDrv**, **dosFsLib**, **rt11FsLib**, **rawFsLib**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## *ioDefPathGet()*

**NAME** *ioDefPathGet()* – get the current default path

**SYNOPSIS**

```
void ioDefPathGet
(
 char * pathname /* where to return the name */
)
```

**DESCRIPTION** This routine copies the name of the current default path to *pathname*. The parameter *pathname* should be **MAX\_FILENAME\_LENGTH** characters long.

**RETURNS** N/A

**SEE ALSO** **ioLib**, *ioDefPathSet()*, *chdir()*, *getcwd()*

---

## *ioDefPathSet()*

**NAME** *ioDefPathSet()* – set the current default path

**SYNOPSIS**

```
STATUS ioDefPathSet
(
 char * name /* name of the new default device and path */
)
```

***ioGlobalStdGet()***

- DESCRIPTION** This routine sets the default I/O path. All relative pathnames specified to the I/O system will be prepended with this pathname. This pathname must be an absolute pathname, i.e., *name* must begin with an existing device name.
- RETURNS** OK, or ERROR if the first component of the pathname is not an existing device.
- SEE ALSO** *ioLib*, *ioDefPathGet()*, *chdir()*, *getcwd()*

---

## ***ioGlobalStdGet()***

- NAME** *ioGlobalStdGet()* – get the file descriptor for global standard input/output/error
- SYNOPSIS**
- ```
int ioGlobalStdGet
(
    int stdFd /* std input (0), output (1), or error (2) */
)
```
- DESCRIPTION** This routine returns the current underlying file descriptor for global standard input, output, and error.
- RETURNS** The underlying global file descriptor, or ERROR if *stdFd* is not 0, 1, or 2.
- SEE ALSO** *ioLib*, *ioGlobalStdSet()*, *ioTaskStdGet()*

ioGlobalStdSet()

- NAME** *ioGlobalStdSet()* – set the file descriptor for global standard input/output/error
- SYNOPSIS**
- ```
void ioGlobalStdSet
(
 int stdFd, /* std input (0), output (1), or error (2) */
 int newFd /* new underlying file descriptor */
)
```
- DESCRIPTION** This routine changes the assignment of a specified global standard file descriptor *stdFd* (0, 1, or 2) to the specified underlying file descriptor *newFd*. *newFd* should be a file descriptor open to the desired device or file. All tasks will use this new assignment when doing I/O to *stdFd*, unless they have specified a task-specific standard file descriptor (see *ioTaskStdSet()*). If *stdFd* is not 0, 1, or 2, this routine has no effect.

**RETURNS** N/A

**SEE ALSO** *ioLib*, *ioGlobalStdGet()*, *ioTaskStdSet()*

---

## *iOlicomEndLoad()*

**NAME** *iOlicomEndLoad()* – initialize the driver and device

**SYNOPSIS**

```
END_OBJ * iOlicomEndLoad
(
 char * initString /* String to be parsed by the driver. */
)
```

**DESCRIPTION** This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the *initString*.

This routine can be called in two modes. If it is called with an empty, but allocated string then it places the name of this device (i.e. *oli*) into the *initString* and returns 0.

If the string is allocated then the routine attempts to perform its load functionality.

**RETURNS** An END object pointer or NULL on error or 0 and the name of the device if the *initString* was NULL.

**SEE ALSO** *iOlicomEnd*

---

## *iOlicomIntHandle()*

**NAME** *iOlicomIntHandle()* – interrupt service for card interrupts

**SYNOPSIS**

```
void iolicomIntHandle
(
 END_DEVICE * pDrvCtrl /* pointer to END_DEVICE structure */
)
```

**DESCRIPTION** This routine is called when an interrupt has been detected from the Olicom card.

**RETURNS** N/A.

**SEE ALSO** *iOlicomEnd*

---

## *ioMmuMicroSparcInit()*

**NAME** *ioMmuMicroSparcInit()* – initialize the microSparc I/II I/O MMU data structures

**SYNOPSIS**

```
STATUS ioMmuMicroSparcInit
(
 void * physBase, /* first valid DMA physical address */
 UINT range /* range covered by I/O Page Table */
)
```

**DESCRIPTION** This routine initializes the I/O MMU for S-Bus DMA with the TMS390S10 and Mb86904. This function is executed after the VxWorks kernel is initialized. The memory allocated for the **ioPage** tables is write protected and cache inhibited only if one of the MMU libraries (**vmBaseLib** or **vmLib**) is initialized. It has been implemented this way because boot ROMs do not initialize the MMU library in **bootConfig.c**; instead, they initialize the MMU separately from **romInit.s**.

**RETURNS** OK, or ERROR if unable to satisfy request.

**SEE ALSO** *ioMmuMicroSparcLib*, *ioMmuMicroSparcMap()*

---

## *ioMmuMicroSparcMap()*

**NAME** *ioMmuMicroSparcMap()* – map the I/O MMU for microSparc I/II (TMS390S10/MB86904)

**SYNOPSIS**

```
STATUS ioMmuMicroSparcMap
(
 UINT dvmaAdrs, /* ioDvma virtual address to map */
 void * physBase, /* physical address to add */
 UINT size /* size to map */
)
```

**DESCRIPTION** This routine maps the specified amount of memory (*size*), starting at the specified **ioDvma** virtual address (*dvmaAdrs*), to the specified physical base (*physBase*).

Do not call *ioMmuMicroSparcMap()* without first calling the initialization routine *ioMmuMicroSparcInit()*, because this routine depends on the data structures initialized there. The *ioMmuMicroSparcMap()* routine checks that the I/O MMU range specified at initialization is sufficient for the size of the memory being mapped. The physical base specified should be on a page boundary. Similarly, the size of the memory being mapped must be a multiple of the page size.



**RETURNS** OK, or ERROR if unable to satisfy request.

**SEE ALSO** *ioMmuMicroSparcLib*, *ioMmuMicroSparcInit()*

---

## *iosDevAdd()*

**NAME** *iosDevAdd()* – add a device to the I/O system

**SYNOPSIS**

```
STATUS iosDevAdd
(
 DEV_HDR * pDevHdr, /* pointer to device's structure */
 char * name, /* name of device */
 int drvnum /* no. of servicing driver, returned by */
)
```

**DESCRIPTION** This routine adds a device to the I/O system device list, making the device available for subsequent *open()* and *creat()* calls.

The parameter *pDevHdr* is a pointer to a device header, **DEV\_HDR** (defined in **iosLib.h**), which is used as the node in the device list. Usually this is the first item in a larger device structure for the specific device type. The parameters *name* and *drnum* are entered in *pDevHdr*.

**RETURNS** OK, or ERROR if there is already a device with the specified name.

**SEE ALSO** *iosLib*

---

## *iosDevDelete()*

**NAME** *iosDevDelete()* – delete a device from the I/O system

**SYNOPSIS**

```
void iosDevDelete
(
 DEV_HDR * pDevHdr /* pointer to device's structure */
)
```

**DESCRIPTION** This routine deletes a device from the I/O system device list, making it unavailable to subsequent *open()* or *creat()* calls. No interaction with the driver occurs, and any file descriptors open on the device or pending operations are unaffected.

***iosDevFind()***

If the device was never added to the device list, unpredictable results may occur.

**RETURNS** N/A

**SEE ALSO** **iosLib**

***iosDevFind()***

**NAME** *iosDevFind()* – find an I/O device in the device list

**SYNOPSIS**

```
DEV_HDR *iosDevFind
(
 char * name, /* name of the device */
 char * *pNameTail /* where to put ptr to tail of name */
)
```

**DESCRIPTION** This routine searches the device list for a device whose name matches the first portion of *name*. If a device is found, *iosDevFind()* sets the character pointer pointed to by *pNameTail* to point to the first character in *name*, following the portion which matched the device name. It then returns a pointer to the device. If the routine fails, it returns a pointer to the default device (that is, the device where the current working directory is mounted) and sets *pNameTail* to point to the beginning of *name*. If there is no default device, *iosDevFind()* returns NULL.

**RETURNS** A pointer to the device header, or NULL if the device is not found.

**SEE ALSO** **iosLib**

***iosDevShow()***

**NAME** *iosDevShow()* – display the list of devices in the system

**SYNOPSIS** `void iosDevShow (void)`

**DESCRIPTION** This routine displays a list of all devices in the device list.

**RETURNS** N/A

**SEE ALSO** **iosShow**, **devs()**, *VxWorks Programmer's Guide: I/O System*, **windsh**, *Tornado User's Guide: Shell*

---

## *iosDrvInstall()*

**NAME** *iosDrvInstall()* – install an I/O driver

**SYNOPSIS**

```
int iosDrvInstall
(
 FUNCPTTR pCreate, /* pointer to driver create function */
 FUNCPTTR pDelete, /* pointer to driver delete function */
 FUNCPTTR pOpen, /* pointer to driver open function */
 FUNCPTTR pClose, /* pointer to driver close function */
 FUNCPTTR pRead, /* pointer to driver read function */
 FUNCPTTR pWrite, /* pointer to driver write function */
 FUNCPTTR pIoctl /* pointer to driver ioctl function */
)
```

**DESCRIPTION** This routine should be called once by each I/O driver. It hooks up the various I/O service calls to the driver service routines, assigns the driver a number, and adds the driver to the driver table.

**RETURNS** The driver number of the new driver, or ERROR if there is no room for the driver.

**SEE ALSO** *iosLib*

---

## *iosDrvRemove()*

**NAME** *iosDrvRemove()* – remove an I/O driver

**SYNOPSIS**

```
STATUS iosDrvRemove
(
 int drvnum, /* no. of driver to remove, returned by iosDrvInstall() */
 BOOL forceClose /* if TRUE, force closure of open files */
)
```

**DESCRIPTION** This routine removes an I/O driver (added by *iosDrvInstall()*) from the driver table.

**RETURNS** OK, or ERROR if the driver has open files.

**SEE ALSO** *iosLib*, *iosDrvInstall()*

---

## ***iosDrvShow()***

|                    |                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>iosDrvShow()</i> – display a list of system drivers                                                              |
| <b>SYNOPSIS</b>    | <code>void iosDrvShow (void)</code>                                                                                 |
| <b>DESCRIPTION</b> | This routine displays a list of all drivers in the driver list.                                                     |
| <b>RETURNS</b>     | N/A                                                                                                                 |
| <b>SEE ALSO</b>    | <b>iosShow</b> , <i>VxWorks Programmer's Guide: I/O System</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i> |

---

## ***iosFdShow()***

|                    |                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>iosFdShow()</i> – display a list of file descriptor names in the system                                                           |
| <b>SYNOPSIS</b>    | <code>void iosFdShow (void)</code>                                                                                                   |
| <b>DESCRIPTION</b> | This routine displays a list of all file descriptors in the system.                                                                  |
| <b>RETURNS</b>     | N/A                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>iosShow</b> , <i>ioctl()</i> , <i>VxWorks Programmer's Guide: I/O System</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i> |

---

## ***iosFdValue()***

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>iosFdValue()</i> – validate an open file descriptor and return the driver-specific value     |
| <b>SYNOPSIS</b>    | <pre>int iosFdValue (     int fd /* file descriptor to check */ )</pre>                         |
| <b>DESCRIPTION</b> | This routine checks to see if a file descriptor is valid and returns the driver-specific value. |
| <b>RETURNS</b>     | The driver-specific value, or ERROR if the file descriptor is invalid.                          |

SEE ALSO *iosLib*

---

## *iosInit()*

**NAME** *iosInit()* – initialize the I/O system

**SYNOPSIS**

```
STATUS iosInit
(
 int max_drivers, /* maximum number of drivers allowed */
 int max_files, /* max number of files allowed open at once */
 char * nullDevName /* name of the null device (bit bucket) */
)
```

**DESCRIPTION** This routine initializes the I/O system. It must be called before any other I/O system routine.

**RETURNS** OK, or ERROR if memory is insufficient.

**SEE ALSO** *iosLib*

---

## *iosShowInit()*

**NAME** *iosShowInit()* – initialize the I/O system show facility

**SYNOPSIS** `void iosShowInit (void)`

**DESCRIPTION** This routine links the I/O system show facility into the VxWorks system. It is called automatically when `INCLUDE_SHOW_ROUTINES` is defined in `configAll.h`.

**RETURNS** N/A

**SEE ALSO** *iosShow*

---

## ***ioTaskStdGet()***

|                    |                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ioTaskStdGet()</i> – get the file descriptor for task standard input/output/error                                                              |
| <b>SYNOPSIS</b>    | <pre>int ioTaskStdGet (     int taskId, /* ID of desired task (0 = self) */     int stdFd   /* std input (0), output (1), or error (2) */ )</pre> |
| <b>DESCRIPTION</b> | This routine returns the current underlying file descriptor for task-specific standard input, output, and error.                                  |
| <b>RETURNS</b>     | The underlying file descriptor, or ERROR if <i>stdFd</i> is not 0, 1, or 2, or the routine is called at interrupt level.                          |
| <b>SEE ALSO</b>    | <i>ioLib</i> , <i>ioGlobalStdGet()</i> , <i>ioTaskStdSet()</i>                                                                                    |

---

## ***ioTaskStdSet()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ioTaskStdSet()</i> – set the file descriptor for task standard input/output/error                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>void ioTaskStdSet (     int taskId, /* task whose std fd is to be set (0 = self) */     int stdFd,  /* std input (0), output (1), or error (2) */     int newFd   /* new underlying file descriptor */ )</pre>                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | This routine changes the assignment of a specified task-specific standard file descriptor <i>stdFd</i> (0, 1, or 2) to the specified underlying file descriptor <i>newFd</i> . <i>newFd</i> should be a file descriptor open to the desired device or file. The calling task will use this new assignment when doing I/O to <i>stdFd</i> , instead of the system-wide global assignment which is used by default. If <i>stdFd</i> is not 0, 1, or 2, this routine has no effect. |
| <b>NOTE</b>        | This routine has no effect if it is called at interrupt level.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <i>ioLib</i> , <i>ioGlobalStdGet()</i> , <i>ioTaskStdGet()</i>                                                                                                                                                                                                                                                                                                                                                                                                                   |

---

## *ipAttach()*

**NAME** *ipAttach()* – a generic attach routine for the TCP/IP network stack

**SYNOPSIS**

```
int ipAttach
(
 int unit, /* Unit number */
 char * pDevice /* Device name (i.e. ln, ei etc.). */
)
```

**DESCRIPTION** This routine takes the unit number and device name of an END driver (e.g., "ln0", "ei0", etc.) and attaches the TCP/IP stack to the MUX. If completed successfully, the IP protocol will begin receiving packets from that driver.

**RETURNS** OK or ERROR

**SEE ALSO** *ipProto*

---

## *ipDetach()*

**NAME** *ipDetach()* – a generic detach routine for the TCP/IP network stack

**SYNOPSIS**

```
STATUS ipDetach
(
 int unit, /* Unit number */
 char * pDevice /* Device name (i.e. ln, ei etc.). */
)
```

**DESCRIPTION** This routine removes the TCP/IP stack from the MUX. If completed successfully, the IP protocol will no longer receive packets from the named END driver.

**RETURNS** OK or ERROR

**SEE ALSO** *ipProto*

---

## *ipFilterHookAdd()*

**NAME** *ipFilterHookAdd()* – add a routine to receive all internet protocol packets

**SYNOPSIS**

```
STATUS ipFilterHookAdd
(
 FUNCPTR ipFilterHook /* routine to receive raw ip packets */
)
```

**DESCRIPTION** This routine adds a hook routine that will be called for every IP packet that is received. The calling sequence of the filter hook routine is:

```
BOOL ipFilterHook
(
 struct ifnet *pIf, /* interface packet was received on */
 struct mbuf **pPtrMbuf, /* pointer to pointer to an mbuf chain */
 struct ip **pPtrIpHdr, /* pointer to pointer to ip header */
 int ipHdrLen, /* ip packet header length */
)
```

The hook routine should return TRUE if it has handled the input packet and no further action should be taken with it. If returning TRUE the *ipFilterHook* is responsible for freeing the mbuf chain by calling *m\_freem(\*pPtrMbuf)*. It should return FALSE if it has not handled the *ipFilterHook* and normal processing (e.g., Internet) should take place.

The packet is in a mbuf chain of which a pointer to a pointer is passed as one of the arguments. The pointer to the mbuf should be accessed by dereferencing the pointer to pointer, *pPtrMbuf*. This mbuf chain will be reused upon return from the hook. If the hook routine needs to retain the input packet, it should copy it elsewhere. by using the macro *copy\_from\_mbufs* (buffer, \**pPtrMbuf*, len). *copy\_from\_mbufs* is defined "**net/mbuf.h**"

*pPtrIpHdr* is a pointer to a pointer to a IP header. The pointer to the ip header is obtained by dereferencing *pPtrIpHdr*. The ip header is used to examine and process the fields in the ip header. The fields *ip\_len*, *ip\_id* and *ip\_offset* in the ip header are converted to the host byte order from the network byte order before a packet is handed to the filter hook.

The *pPtrMbuf* and *pPtrIpHdr* are reused upon return from the hook if it is returning FALSE.

Normally you will not be needing to modify *pPtrMbuf* or the *pPtrIpHdr*.

**RETURNS** OK, always.

**SEE ALSO** *ipFilterLib*



---

## *ipFilterHookDelete()*

**NAME** *ipFilterHookDelete()* – delete a ip filter hook routine

**SYNOPSIS** `void ipFilterHookDelete (void)`

**DESCRIPTION** This routine deletes an IP filter hook.

**SEE ALSO** `ipFilterLib`

---

## *ipFilterLibInit()*

**NAME** *ipFilterLibInit()* – initialize ip filter facility

**SYNOPSIS** `void ipFilterLibInit (void)`

**DESCRIPTION** This routine links the ip filter facility into the VxWorks system. These routines are included automatically if `INCLUDE_IP_FILTER` is defined in `configAll.h`.

**RETURNS** N/A

**SEE ALSO** `ipFilterLib`

---

## *ipstatShow()*

**NAME** *ipstatShow()* – display IP statistics

**SYNOPSIS**

```
void ipstatShow
(
 BOOL zero /* TRUE = reset statistics to 0 */
)
```

**DESCRIPTION** This routine displays detailed statistics for the IP protocol.

**RETURNS** N/A

**SEE ALSO** `netShow`

---

## *rint()*

|                      |                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>rint()</i> – convert a double-precision value to an integer                                                    |
| <b>SYNOPSIS</b>      | <pre>int rint (     double x /* argument */ )</pre>                                                               |
| <b>DESCRIPTION</b>   | This routine converts a double-precision value <i>x</i> to an integer using the selected IEEE rounding direction. |
| <b>CAVEAT</b>        | The rounding direction is not pre-selectable and is fixed for round-to-the-nearest.                               |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                     |
| <b>RETURNS</b>       | The integer representation of <i>x</i> .                                                                          |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                                   |

---

## *rintf()*

|                      |                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>rintf()</i> – convert a single-precision value to an integer                                                   |
| <b>SYNOPSIS</b>      | <pre>int rintf (     float x /* argument */ )</pre>                                                               |
| <b>DESCRIPTION</b>   | This routine converts a single-precision value <i>x</i> to an integer using the selected IEEE rounding direction. |
| <b>CAVEAT</b>        | The rounding direction is not pre-selectable and is fixed as round-to-the-nearest.                                |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                     |
| <b>RETURNS</b>       | The integer representation of <i>x</i> .                                                                          |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                                   |

---

## *iround()*

|                      |                                                                                     |
|----------------------|-------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>iround()</i> – round a number to the nearest integer                             |
| <b>SYNOPSIS</b>      | <pre>int iround (     double x /* argument */ )</pre>                               |
| <b>DESCRIPTION</b>   | This routine rounds a double-precision value <i>x</i> to the nearest integer value. |
| <b>NOTE</b>          | If <i>x</i> is spaced evenly between two integers, it returns the even integer.     |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                       |
| <b>RETURNS</b>       | The integer nearest to <i>x</i> .                                                   |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                     |

---

## *iroundf()*

|                      |                                                                                     |
|----------------------|-------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>iroundf()</i> – round a number to the nearest integer                            |
| <b>SYNOPSIS</b>      | <pre>int iroundf (     float x /* argument */ )</pre>                               |
| <b>DESCRIPTION</b>   | This routine rounds a single-precision value <i>x</i> to the nearest integer value. |
| <b>NOTE</b>          | If <i>x</i> is spaced evenly between two integers, the even integer is returned.    |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                       |
| <b>RETURNS</b>       | The integer nearest to <i>x</i> .                                                   |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                     |

## *isalnum()*

|                      |                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>isalnum()</i> – test whether a character is alphanumeric (ANSI)                                              |
| <b>SYNOPSIS</b>      | <pre>int isalnum (     int c /* character to test */ )</pre>                                                    |
| <b>DESCRIPTION</b>   | This routine tests whether <i>c</i> is a character for which <i>isalpha()</i> or <i>isdigit()</i> returns true. |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                                                            |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is alphanumeric.                                                               |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                                                          |

---

## *isalpha()*

|                      |                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>isalpha()</i> – test whether a character is a letter (ANSI)                                                  |
| <b>SYNOPSIS</b>      | <pre>int isalpha (     int c /* character to test */ )</pre>                                                    |
| <b>DESCRIPTION</b>   | This routine tests whether <i>c</i> is a character for which <i>isupper()</i> or <i>islower()</i> returns true. |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                                                            |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is a letter.                                                                   |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                                                          |

---

## *isatty()*

|                    |                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>isatty()</i> – return whether the underlying driver is a tty device                              |
| <b>SYNOPSIS</b>    | <pre>BOOL isatty (     int fd /* file descriptor to check */ )</pre>                                |
| <b>DESCRIPTION</b> | This routine simply invokes the <i>ioctl()</i> function FIOISATTY on the specified file descriptor. |
| <b>RETURNS</b>     | TRUE, or FALSE if the driver does not indicate a tty device.                                        |
| <b>SEE ALSO</b>    | ioLib                                                                                               |

---

## *iscntrl()*

|                      |                                                                           |
|----------------------|---------------------------------------------------------------------------|
| <b>NAME</b>          | <i>iscntrl()</i> – test whether a character is a control character (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int iscntrl (     int c /* character to test */ )</pre>              |
| <b>DESCRIPTION</b>   | This routine tests whether <i>c</i> is a control character.               |
| <b>INCLUDE FILES</b> | ctype.h                                                                   |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is a control character.                  |
| <b>SEE ALSO</b>      | ansiCtype                                                                 |

## ***isdigit()***

|                      |                                                                       |
|----------------------|-----------------------------------------------------------------------|
| <b>NAME</b>          | <i>isdigit()</i> – test whether a character is a decimal digit (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int isdigit (     int c /* character to test */ )</pre>          |
| <b>DESCRIPTION</b>   | This routine tests whether <i>c</i> is a decimal-digit character.     |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                  |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is a decimal digit.                  |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                |

---

## ***isgraph()***

|                      |                                                                                                                             |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>isgraph()</i> – test whether a character is a printing, non-white-space character (ANSI)                                 |
| <b>SYNOPSIS</b>      | <pre>int isgraph (     int c /* character to test */ )</pre>                                                                |
| <b>DESCRIPTION</b>   | This routine returns true if <i>c</i> is a printing character, and not a character for which <i>isspace()</i> returns true. |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                                                                        |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is a printable, non-white-space character.                                                 |
| <b>SEE ALSO</b>      | <code>ansiCtype</code> , <i>isspace()</i>                                                                                   |

---

## *islower()*

|                      |                                                                           |
|----------------------|---------------------------------------------------------------------------|
| <b>NAME</b>          | <i>islower()</i> – test whether a character is a lower-case letter (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int islower (     int c /* character to test */ )</pre>              |
| <b>DESCRIPTION</b>   | This routine tests whether <i>c</i> is a lower-case letter.               |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                      |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is a lower-case letter.                  |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                    |

---

## *isprint()*

|                      |                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>isprint()</i> – test whether a character is printable, including the space character (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int isprint (     int c /* character to test */ )</pre>                                   |
| <b>DESCRIPTION</b>   | This routine returns true if <i>c</i> is a printing character or the space character.          |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                                           |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is printable, including the space character.                  |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                                         |

## *ispunct()*

|                      |                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>ispunct()</i> – test whether a character is punctuation (ANSI)                                                                                  |
| <b>SYNOPSIS</b>      | <pre>int ispunct (     int c /* character to test */ )</pre>                                                                                       |
| <b>DESCRIPTION</b>   | This routine tests whether a character is punctuation, i.e., a printing character for which neither <i>isspace()</i> nor <i>isalnum()</i> is true. |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                                                                                               |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is a punctuation character.                                                                                       |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                                                                                             |

---

## *isspace()*

|                      |                                                                                                                                                                                                                                                                                                                                                         |       |    |                |    |              |    |                 |    |          |    |           |    |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|----|----------------|----|--------------|----|-----------------|----|----------|----|-----------|----|
| <b>NAME</b>          | <i>isspace()</i> – test whether a character is a white-space character (ANSI)                                                                                                                                                                                                                                                                           |       |    |                |    |              |    |                 |    |          |    |           |    |
| <b>SYNOPSIS</b>      | <pre>int isspace (     int c /* character to test */ )</pre>                                                                                                                                                                                                                                                                                            |       |    |                |    |              |    |                 |    |          |    |           |    |
| <b>DESCRIPTION</b>   | This routine tests whether a character is a standard white-space character, as follows:<br><table><tr><td>space</td><td>""</td></tr><tr><td>horizontal tab</td><td>\t</td></tr><tr><td>vertical tab</td><td>\v</td></tr><tr><td>carriage return</td><td>\r</td></tr><tr><td>new-line</td><td>\n</td></tr><tr><td>form-feed</td><td>\f</td></tr></table> | space | "" | horizontal tab | \t | vertical tab | \v | carriage return | \r | new-line | \n | form-feed | \f |
| space                | ""                                                                                                                                                                                                                                                                                                                                                      |       |    |                |    |              |    |                 |    |          |    |           |    |
| horizontal tab       | \t                                                                                                                                                                                                                                                                                                                                                      |       |    |                |    |              |    |                 |    |          |    |           |    |
| vertical tab         | \v                                                                                                                                                                                                                                                                                                                                                      |       |    |                |    |              |    |                 |    |          |    |           |    |
| carriage return      | \r                                                                                                                                                                                                                                                                                                                                                      |       |    |                |    |              |    |                 |    |          |    |           |    |
| new-line             | \n                                                                                                                                                                                                                                                                                                                                                      |       |    |                |    |              |    |                 |    |          |    |           |    |
| form-feed            | \f                                                                                                                                                                                                                                                                                                                                                      |       |    |                |    |              |    |                 |    |          |    |           |    |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                                                                                                                                                                                                                                                                                                    |       |    |                |    |              |    |                 |    |          |    |           |    |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is a space, tab, carriage return, new-line, or form-feed character.                                                                                                                                                                                                                                                    |       |    |                |    |              |    |                 |    |          |    |           |    |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                                                                                                                                                                                                                                                                                                  |       |    |                |    |              |    |                 |    |          |    |           |    |



---

## *isupper()*

|                      |                                                                            |
|----------------------|----------------------------------------------------------------------------|
| <b>NAME</b>          | <i>isupper()</i> – test whether a character is an upper-case letter (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int isupper (     int c /* character to test */ )</pre>               |
| <b>DESCRIPTION</b>   | This routine tests whether <i>c</i> is an upper-case letter.               |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                       |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is an upper-case letter.                  |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                     |

---

## *isxdigit()*

|                      |                                                                            |
|----------------------|----------------------------------------------------------------------------|
| <b>NAME</b>          | <i>isxdigit()</i> – test whether a character is a hexadecimal digit (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int isxdigit (     int c /* character to test */ )</pre>              |
| <b>DESCRIPTION</b>   | This routine tests whether <i>c</i> is a hexadecimal-digit character.      |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                       |
| <b>RETURNS</b>       | Non-zero if and only if <i>c</i> is a hexadecimal digit.                   |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                     |

***kernelInit()***

---

***kernelInit()*****NAME** *kernelInit()* – initialize the kernel

**SYNOPSIS**

```
void kernelInit
(
 FUNCPTR rootRtn, /* user start-up routine */
 unsigned rootMemSize, /* memory for TCB and root stack */
 char * pMemPoolStart, /* beginning of memory pool */
 char * pMemPoolEnd, /* end of memory pool */
 unsigned intStackSize, /* interrupt stack size */
 int lockOutLevel /* interrupt lock-out level (1-7) */
)
```

**DESCRIPTION** This routine initializes and starts the kernel. It should be called only once. The parameter *rootRtn* specifies the entry point of the user's start-up code that subsequently initializes system facilities (i.e., the I/O system, network). Typically, *rootRtn* is set to *usrRoot()*.

Interrupts are enabled for the first time after *kernelInit()* exits. VxWorks will not exceed the specified interrupt lock-out level during any of its brief uses of interrupt locking as a means of mutual exclusion.

The system memory partition is initialized by *kernelInit()* with the size set by *pMemPoolStart* and *pMemPoolEnd*. Architectures that support a separate interrupt stack allocate a portion of memory for this purpose, of *intStackSize* bytes starting at *pMemPoolStart*.

**RETURNS** N/A**SEE ALSO** *kernelLib*, *intLockLevelSet()*

---

***kernelTimeSlice()*****NAME** *kernelTimeSlice()* – enable round-robin selection

**SYNOPSIS**

```
STATUS kernelTimeSlice
(
 int ticks /* time-slice in ticks or 0 to disable round-robin */
)
```

|                    |                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine enables round-robin selection among tasks of same priority and sets the system time-slice to <i>ticks</i> . Round-robin scheduling is disabled by default. A time-slice of zero ticks disables round-robin scheduling. For more information about round-robin scheduling, see the manual entry for <b>kernelLib</b> . |
| <b>RETURNS</b>     | OK, always.                                                                                                                                                                                                                                                                                                                        |
| <b>SEE ALSO</b>    | <b>kernelLib</b>                                                                                                                                                                                                                                                                                                                   |

---

## *kernelVersion()*

|                    |                                                                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>kernelVersion()</i> – return the kernel revision string                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <code>char *kernelVersion (void)</code>                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | This routine returns a string which contains the current revision of the kernel. The string is of the form "WIND version x.y", where "x" corresponds to the kernel major revision, and "y" corresponds to the kernel minor revision. |
| <b>RETURNS</b>     | A pointer to a string of format "WIND version x.y".                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>kernelLib</b>                                                                                                                                                                                                                     |

---

## *kill()*

|                    |                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>kill()</i> – send a signal to a task (POSIX)                                                              |
| <b>SYNOPSIS</b>    | <pre>int kill (     int tid, /* task to send signal to */     int signo /* signal to send to task */ )</pre> |
| <b>DESCRIPTION</b> | This routine sends a signal <i>signo</i> to the task specified by <i>tid</i> .                               |
| <b>RETURNS</b>     | OK (0), or ERROR (-1) if the task ID or signal number is invalid.                                            |
| <b>ERRNO</b>       | EINVAL                                                                                                       |
| <b>SEE ALSO</b>    | <b>sigLib</b>                                                                                                |

**l()**

---

**l()****NAME** `l()` – disassemble and display a specified number of instructions

**SYNOPSIS**

```
void l
(
 INSTR * addr, /* address of first instruction to disassemble if 0, */
 /* from the last instruction disassembled on the */
 /* call to l */
 int count /* number of instruction to disassemble */
 /* if 0, use the same the last call to l */
)
```

**DESCRIPTION** This routine disassembles a specified number of instructions and displays them on standard output. If the address of an instruction is entered in the system symbol table, the symbol will be displayed as a label for that instruction. Also, addresses in the opcode field of instructions will be displayed symbolically.

To execute, enter:

```
-> l [address [,count]]
```

If *address* is omitted or zero, disassembly continues from the previous address. If *count* is omitted or zero, the last specified count is used (initially 10). As with all values entered via the shell, the address may be typed symbolically.

**RETURNS** N/A**SEE ALSO** `dbgLib`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

**l0()****NAME** `l0()` – return the contents of register **I0** (also **I1** – **I7**) (SPARC)

**SYNOPSIS**

```
int l0
(
 int taskId /* task ID, 0 means default task */
)
```

**DESCRIPTION** This command extracts the contents of local register **I0** from the TCB of a specified task. If *taskId* is omitted or 0, the current default task is assumed.

Similar routines are provided for all local registers (10 – 17): *l0()* – *l7()*.

**RETURNS** The contents of register 10 (or the requested register).

**SEE ALSO** *dbgArchLib*, *VxWorks Programmer's Guide: Target Shell*

## *labs()*

**NAME** *labs()* – compute the absolute value of a **long** (ANSI)

**SYNOPSIS**

```
long labs
(
 long i /* long for which to return absolute value */
)
```

**DESCRIPTION** This routine computes the absolute value of a specified **long**. If the result cannot be represented, the behavior is undefined. This routine is equivalent to *abs()*, except that the argument and return value are all of type **long**.

**INCLUDE FILES** *stdlib.h*

**RETURNS** The absolute value of *i*.

**SEE ALSO** *ansiStdlib*

## *ld()*

**NAME** *ld()* – load an object module into memory

**SYNOPSIS**

```
MODULE_ID ld
(
 int syms, /* -1, 0, or 1 */
 BOOL noAbort, /* TRUE = don't abort script on error */
 char * name /* name of object module, NULL = standard input */
)
```

**DESCRIPTION** This command loads an object module from a file or from standard input. The object module must be in UNIX *a.out* format. External references in the module are resolved during loading. The *syms* parameter determines how symbols are loaded; possible values:

***ldexp()***

- 0 – Add global symbols to the system symbol table.
- 1 – Add global and local symbols to the system symbol table.
- 1 – Add no symbols to the system symbol table.

If there is an error during loading (e.g., externals undefined, too many symbols, etc.), then *shellScriptAbort()* is called to stop any script that this routine was called from. If *noAbort* is TRUE, errors are noted but ignored.

The normal way of using *ld()* is to load all symbols (*syms* = 1) during debugging and to load only global symbols later.

**EXAMPLE**

The following example loads the **a.out** file **module** from the default file device into memory, and adds any global symbols to the symbol table:

```
-> ld <module
```

This example loads **test.o** with all symbols:

```
-> ld 1,0,"test.o"
```

**RETURNS**

MODULE\_ID, or NULL if there are too many symbols, the object file format is invalid, or there is an error reading the file.

**SEE ALSO**

**usrLib**, **loadLib**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

***ldexp()*****NAME**

*ldexp()* – multiply a number by an integral power of 2 (ANSI)

**SYNOPSIS**

```
double ldexp
(
 double v, /* a floating point number */
 int xexp /* exponent */
)
```

**DESCRIPTION**

This routine multiplies a floating-point number by an integral power of 2. A range error may occur.

**INCLUDE FILES**

**math.h**

**RETURNS**

The double-precision value of *v* times 2 to the power of *xexp*.

**SEE ALSO**

**ansiMath**

---

## *ldiv()*

|                      |                                                                                                                                                                                                                                                                                                               |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>ldiv()</i> – compute the quotient and remainder of the division (ANSI)                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>      | <pre>ldiv_t ldiv (     long numer, /* numerator */     long denom /* denominator */ )</pre>                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>   | <p>This routine computes the quotient and remainder of <i>numer/denom</i>. This routine is similar to <i>div()</i>, except that the arguments and the elements of the returned structure are all of type <b>long</b>.</p> <p>This routine is not reentrant. For a reentrant version, see <i>ldiv_r()</i>.</p> |
| <b>INCLUDE FILES</b> | <b>stdlib.h</b>                                                                                                                                                                                                                                                                                               |
| <b>RETURNS</b>       | A structure of type <b>ldiv_t</b> , containing both the quotient and the remainder.                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>      | <b>ansiStdlib</b>                                                                                                                                                                                                                                                                                             |

---

## *ldiv\_r()*

|                      |                                                                                                                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>ldiv_r()</i> – compute a quotient and remainder (reentrant)                                                                                                                                                                   |
| <b>SYNOPSIS</b>      | <pre>void ldiv_r (     long    numer,      /* numerator */     long    denom,      /* denominator */     ldiv_t * divStructPtr /* ldiv_t structure */ )</pre>                                                                    |
| <b>DESCRIPTION</b>   | <p>This routine computes the quotient and remainder of <i>numer/denom</i>. The quotient and remainder are stored in the <b>ldiv_t</b> structure <b>divStructPtr</b>. This routine is the reentrant version of <i>ldiv()</i>.</p> |
| <b>INCLUDE FILES</b> | <b>stdlib.h</b>                                                                                                                                                                                                                  |
| <b>RETURNS</b>       | N/A                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>      | <b>ansiStdlib</b>                                                                                                                                                                                                                |

## **ledClose()**

|                    |                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ledClose()</i> – discard the line-editor ID                                                                           |
| <b>SYNOPSIS</b>    | <pre>STATUS ledClose (     int led_id /* ID returned by ledOpen */ )</pre>                                               |
| <b>DESCRIPTION</b> | This routine frees resources allocated by <i>ledOpen()</i> . The low-level input/output file descriptors are not closed. |
| <b>RETURNS</b>     | OK.                                                                                                                      |
| <b>SEE ALSO</b>    | <i>ledLib</i> , <i>ledOpen()</i>                                                                                         |

---

## **ledControl()**

|                    |                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ledControl()</i> – change the line-editor ID parameters                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>void ledControl (     int led_id, /* ID returned by ledOpen */     int inFd,   /* new input fd (NONE = no change) */     int outFd,  /* new output fd (NONE = no change) */     int histSize /* new history list size (NONE = no change), */                 /* (0 = display) */ )</pre> |
| <b>DESCRIPTION</b> | This routine changes the input/output file descriptor and the size of the history list.                                                                                                                                                                                                       |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>ledLib</i>                                                                                                                                                                                                                                                                                 |



---

## *ledOpen()*

|                    |                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ledOpen()</i> – create a new line-editor ID                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>int ledOpen (     int inFd,    /* low-level device input fd */     int outFd,   /* low-level device output fd */     int histSize /* size of history list */ )</pre>          |
| <b>DESCRIPTION</b> | This routine creates the ID that is used by <i>ledRead()</i> , <i>ledClose()</i> , and <i>ledControl()</i> . Storage is allocated for up to <i>histSize</i> previously read lines. |
| <b>RETURNS</b>     | The line-editor ID, or ERROR if the routine runs out of memory.                                                                                                                    |
| <b>SEE ALSO</b>    | <i>ledLib</i> , <i>ledRead()</i> , <i>ledClose()</i> , <i>ledControl()</i>                                                                                                         |

---

## *ledRead()*

|                    |                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ledRead()</i> – read a line with line-editing                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>int ledRead (     int led_id, /* ID returned by ledOpen */     char * string, /* where to return line */     int maxBytes /* maximum number of chars to read */ )</pre>                  |
| <b>DESCRIPTION</b> | This routine handles line-editing and history substitutions. If the low-level input file descriptor is not in <i>OPT_LINE</i> mode, only an ordinary <i>read()</i> routine will be performed. |
| <b>RETURNS</b>     | The number of characters read, or EOF.                                                                                                                                                        |
| <b>SEE ALSO</b>    | <i>ledLib</i>                                                                                                                                                                                 |

---

## *lio\_listio()*

**NAME** *lio\_listio()* – initiate a list of asynchronous I/O requests (POSIX)

**SYNOPSIS**

```
int lio_listio
(
 int mode, /* LIO_WAIT or LIO_NOWAIT */
 struct aiocb * list[], /* list of operations */
 int nEnt, /* size of list */
 struct sigevent * pSig /* signal on completion */
)
```

**DESCRIPTION** This routine submits a number of I/O operations (up to `AIO_LISTIO_MAX`) to be performed asynchronously. *list* is a pointer to an array of `aiocb` structures that specify the AIO operations to be performed. The array is of size *nEnt*.

The `aiocb` structure specifies the AIO operation to be performed. Valid entries include `LIO_READ`, `LIO_WRITE`, and `LIO_NOP`. `LIO_READ` corresponds to a call to `aioread()`, `LIO_WRITE` corresponds to a call to `aiowrite()`, and `LIO_NOP` is ignored.

The *mode* argument can be either `LIO_WAIT` or `LIO_NOWAIT`. If *mode* is `LIO_WAIT`, `lio_listio()` does not return until all the AIO operations complete and the *pSig* argument is ignored. If *mode* is `LIO_NOWAIT`, the `lio_listio()` returns as soon as the operations are queued. In this case, if *pSig* is not NULL and the signal number indicated by `pSig>sigev_signo` is not zero, the signal `pSig>sigev_signo` is delivered when all requests have completed.

**RETURNS** OK if requests queued successfully, otherwise ERROR.

**ERRNO** EINVAL, EAGAIN, EIO

**INCLUDE FILES** aio.h

**SEE ALSO** aioPxLib, `aioread()`, `aiowrite()`, `aioreaderror()`, `aioreturn()`.

---

## *listen()*

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>listen()</i> – enable connections to a socket                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS listen (     int s,          /* socket descriptor */     int backlog /* number of connections to queue */ )</pre>                                                                                                                                             |
| <b>DESCRIPTION</b> | This routine enables connections to a socket. It also specifies the maximum number of unaccepted connections that can be pending at one time ( <i>backlog</i> ). After enabling connections with <i>listen()</i> , connections are actually accepted by <i>accept()</i> . |
| <b>RETURNS</b>     | OK, or ERROR if the socket is invalid or unable to listen.                                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | sockLib                                                                                                                                                                                                                                                                   |

---

## *lkAddr()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lkAddr()</i> – list symbols whose values are near a specified value                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>void lkAddr (     unsigned int addr /* address around which to look */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This command lists the symbols in the system symbol table that are near a specified value. The symbols that are displayed include:</p> <ul style="list-style-type: none"><li>– symbols whose values are immediately less than the specified value</li><li>– symbols with the specified value</li><li>– succeeding symbols, until at least 12 symbols have been displayed</li></ul> <p>This command also displays symbols that are local, i.e., symbols found in the system symbol table only because their module was loaded by <i>ld()</i>.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>symLib</b> , <i>symEach()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                                                                                                                                                                                                             |

***lkup()***

---

***lkup()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lkup()</i> – list symbols                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>void lkup (     char * substr /* substring to match */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b> | <p>This command lists all symbols in the system symbol table whose names contain the string <i>substr</i>. If <i>substr</i> is omitted or is 0, a short summary of symbol table statistics is printed. If <i>substr</i> is the empty string (""), all symbols in the table are listed.</p> <p>This command also displays symbols that are local, i.e., symbols found in the system symbol table only because their module was loaded by <i>ld()</i>.</p> <p>By default, <i>lkup()</i> displays 22 symbols at a time. This can be changed by modifying the global variable <b>symLkupPgSz</b>. If this variable is set to 0, <i>lkup()</i> displays all the symbols without interruption.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>symLib</b> , <b>symEach()</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

---

***ll()***

|                    |                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ll()</i> – do a long listing of directory contents                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS ll (     char * dirName /* name of directory to list */ )</pre>                                                                                              |
| <b>DESCRIPTION</b> | <p>This command causes a long listing of a directory's contents to be displayed. It is equivalent to:</p> <pre>-&gt; ls dirName, TRUE</pre>                              |
| <b>NOTE</b>        | When used with <b>netDrv</b> devices (FTP or RSH), <i>ll()</i> does not give directory information. It is equivalent to an <i>ls()</i> call with no long-listing option. |

**RETURNS** OK or ERROR.

**SEE ALSO** *usrLib*, *ls()*, *stat()*, *VxWorks Programmer's Guide: Target Shell*

---

## ***ln97xEndLoad()***

**NAME** *ln97xEndLoad()* – initialize the driver and device

**SYNOPSIS**

```
END_OBJ * ln97xEndLoad
(
 char * initString /* string to be parse by the driver */
)
```

**DESCRIPTION** This routine initializes the driver and the device to the operational state. All of the device-specific parameters are passed in *initString*, which expects a string of the following format:

```
unit:devMemAddr:devIoAddr:pciMemBase:<vecnum:intLvl:memAdrs:memSize:memWidth:csr3b:offset:flags
```

This routine can be called in two modes. If it is called with an empty but allocated string, it places the name of this device (that is, "lnPci") into the *initString* and returns 0.

If the string is allocated and not empty, the routine attempts to load the driver using the values specified in the string.

**RETURNS** An END object pointer, or NULL on error, or 0 and the name of the device if the *initString* was NULL.

**SEE ALSO** *ln97xEnd*

---

## ***ln97xInitParse()***

**NAME** *ln97xInitParse()* – parse the initialization string

**SYNOPSIS**

```
STATUS ln97xInitParse
(
 LN_97X_DRV_CTRL * pDrvCtrl, /* pointer to the control structure */
 char * initString /* initialization string */
)
```

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | <p>Parse the input string. This routine is called from <i>In97xEndLoad()</i> which initializes some values in the driver control structure with the values passed in the initialization string.</p> <p>The initialization string format is:<br/><i>unit:devMemAddr:devIoAddr:pciMemBase:&lt;vecNum:intLvl:memAdrs:memSize:memWidth:csr3b:offset:flags</i></p> <p><i>unit</i><br/>Device unit number, a small integer.</p> <p><i>devMemAddr</i><br/>Device register base memory address</p> <p><i>devIoAddr</i><br/>Device register base IO address</p> <p><i>pciMemBase</i><br/>Base address of PCI memory space</p> <p><i>vecNum</i><br/>Interrupt vector number.</p> <p><i>intLvl</i><br/>Interrupt level.</p> <p><i>memAdrs</i><br/>Memory pool address or NONE.</p> <p><i>memSize</i><br/>Memory pool size or zero.</p> <p><i>memWidth</i><br/>Memory system size, 1, 2, or 4 bytes (optional).</p> <p><i>CSR3</i><br/>Value of CSR3 (for endian-ness mainly)</p> <p><i>offset</i><br/>Offset of starting of data in the device buffers.</p> <p><i>flags</i><br/>Device specific flags, for future use.</p> |
| <b>RETURNS</b>     | OK, or ERROR if any arguments are invalid.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <b>In97xEnd</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

## ***In7990EndLoad()***

- NAME** *In7990EndLoad()* – initialize the driver and device
- SYNOPSIS**
- ```
END_OBJ* In7990EndLoad
(
    char* initString /* string to be parse by the driver */
)
```
- DESCRIPTION**
- This routine initializes the driver and the device to the operational state. All of the device-specific parameters are passed in *initString*, which expects a string of the following format:
- unit:CSR_reg_addr:RAP_reg_addr:int_vector:int_level:shmem_addr:shmem_size:shmem_width*
- This routine can be called in two modes. If it is called with an empty but allocated string, it places the name of this device (that is, "ln") into the *initString* and returns 0.
- If the string is allocated and not empty, the routine attempts to load the driver using the values specified in the string.
- RETURNS** An END object pointer, or NULL on error, or 0 and the name of the device if the *initString* was NULL.
- SEE ALSO** *In7990End*

In7990InitMem()

- NAME** *In7990InitMem()* – initialize memory for Lance chip
- SYNOPSIS**
- ```
STATUS In7990InitMem
(
 LN7990END_DEVICE * pDrvCtrl /* device to be initialized */
)
```
- DESCRIPTION**
- Using data in the control structure, setup and initialize the memory areas needed. If the memory address is not already specified, then allocate cache safe memory.
- RETURNS** OK or ERROR.
- SEE ALSO** *In7990End*

---

## ***In7990InitParse()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b><i>In7990InitParse()</i></b> – parse the initialization string                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>STATUS In7990InitParse (     LN7990END_DEVICE * pDrvCtrl,     char *            initString )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | <p>Parse the input string. Fill in values in the driver control structure. The initialization string format is: <i>unit:csrAdr:rapAdr:vecnum:intLvl:memAdrs:memSize:memWidth:offset:csr3B</i></p> <p><i>unit</i><br/>Device unit number, a small integer.</p> <p><i>csrAdr</i><br/>Address of CSR0 register.</p> <p><i>rapAdr</i><br/>Address of RAP register.</p> <p><i>vecNum</i><br/>Interrupt vector number (used with <i>sysIntConnect()</i> ).</p> <p><i>intLvl</i><br/>Interrupt level.</p> <p><i>memAdrs</i><br/>Memory pool address or NONE.</p> <p><i>memSize</i><br/>Memory pool size or zero.</p> <p><i>memWidth</i><br/>Memory system size, 1, 2, or 4 bytes (optional).</p> <p><i>offset</i><br/>Memory offset for alignment.</p> <p><i>csr3B</i><br/>CSR register 3B control value, normally 0x4 or 0x7.</p> |
| <b>RETURNS</b>     | OK, or ERROR if any arguments are invalid.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>In7990End</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |



---

## ***lnattach()***

**NAME** *lnattach()* – publish the **In** network interface and initialize driver structures

**SYNOPSIS**

```
STATUS lnattach
(
 int unit, /* unit number */
 char * devAdrs, /* LANCE I/O address */
 int ivec, /* interrupt vector */
 int ilevel, /* interrupt level */
 char * memAdrs, /* address of memory pool (-1 = malloc it) */
 ULONG memSize, /* only used if memory pool is NOT malloc()'d */
 int memWidth, /* byte-width of data (-1 = any width) */
 int spare, /* not used */
 int spare2 /* not used */
)
```

**DESCRIPTION** This routine publishes the **In** interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state.

The *memAdrs* parameter can be used to specify the location of the memory that will be shared between the driver and the device. The value **NONE** is used to indicate that the driver should obtain the memory.

The *memSize* parameter is valid only if the *memAdrs* parameter is not set to **NONE**, in which case *memSize* indicates the size of the provided memory region.

The *memWidth* parameter sets the memory pool's data port width (in bytes); if it is **NONE**, any data width is used.

**BUGS** To zero out LANCE data structures, this routine uses *bzero()*, which ignores the *memWidth* specification and uses any size data access to write to memory.

**RETURNS** OK or ERROR.

**SEE ALSO** *if\_in*

**InPciattach()**

---

**InPciattach()**

**NAME** *InPciattach()* – publish the InPci network interface and initialize the driver and device

**SYNOPSIS**

```
STATUS InPciattach
(
 int unit, /* unit number */
 char * devAdrs, /* LANCE I/O address */
 int ivec, /* interrupt vector */
 int ilevel, /* interrupt level */
 char * memAdrs, /* address of memory pool (-1 = malloc it) */
 ULONG memSize, /* used if memory pool is NOT malloc()'d */
 int memWidth, /* byte-width of data (-1 = any width) */
 ULONG pciMemBase, /* memory base as seen from PCI */
 int spare2 /* not used */
)
```

**DESCRIPTION** This routine publishes the In interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state.

The *memAdrs* parameter can be used to specify the location of the memory that will be shared between the driver and the device. The value NONE is used to indicate that the driver should obtain the memory.

The *memSize* parameter is valid only if the *memAdrs* parameter is not set to NONE, in which case *memSize* indicates the size of the provided memory region.

The *memWidth* parameter sets the memory pool's data port width (in bytes); if it is NONE, any data width is used.

**BUGS** To zero out LANCE data structures, this routine uses *bzero()*, which ignores the *memWidth* specification and uses any size data access to write to memory.

**RETURNS** OK or ERROR.

**SEE ALSO** *if\_InPci*

---

## **loadModule()**

|                    |                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>loadModule()</i> – load an object module into memory                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>MODULE_ID loadModule (     int fd,      /* fd of file to load */     int symFlag /* symbols to add to table */                 /* (LOAD_[NO   LOCAL   GLOBAL   ALL]_SYMBOLS) */ ) </pre>                                                                                                                                                    |
| <b>DESCRIPTION</b> | <p>This routine loads an object module from the specified file, and places the code, data, and BSS into memory allocated from the system memory pool.</p> <p>This call is equivalent to <i>loadModuleAt()</i> with NULL for the addresses of text, data, and BSS segments. For more details, see the manual entry for <i>loadModuleAt()</i>.</p> |
| <b>RETURNS</b>     | MODULE_ID, or NULL if the routine cannot read the file, there is not enough memory, or the file format is illegal.                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <i>loadLib</i> , <i>loadModuleAt()</i>                                                                                                                                                                                                                                                                                                           |

---

## **loadModuleAt()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>loadModuleAt()</i> – load an object module into memory                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>MODULE_ID loadModuleAt (     int  fd,      /* fd from which to read module */     int  symFlag, /* symbols to add to table */                 /* (LOAD_[NO   LOCAL   GLOBAL   ALL]_SYMBOLS) */     char * *ppText, /* load text segment at addr pointed to by this ptr, */                 /* load addr via this ptr */     char * *ppData, /* load data segment at addr pointed to by this */                 /* return load addr via this ptr */     char * *ppBss  /* load BSS segment at addr pointed to by this ptr, */                 /* load addr via this ptr */ ) </pre> |
| <b>DESCRIPTION</b> | <p>This routine reads an object module from <i>fd</i>, and loads the code, data, and BSS segments at the specified load addresses in memory set aside by the user using <i>malloc()</i>, or in the</p>                                                                                                                                                                                                                                                                                                                                                                                  |

system memory partition as described below. The module is properly relocated according to the relocation commands in the file. Unresolved externals will be linked to symbols found in the system symbol table. Symbols in the module being loaded can optionally be added to the system symbol table.

#### LINKING UNRESOLVED EXTERNALS

As the module is loaded, any unresolved external references are resolved by looking up the missing symbols in the the system symbol table. If found, those references are correctly linked to the new module. If unresolved external references cannot be found in the system symbol table, then an error message ("undefined symbol: ...") is printed for the symbol, but the loading/linking continues. In this case, NULL will be returned after the module is loaded.

#### ADDING SYMBOLS TO THE SYMBOL TABLE

The symbols defined in the module to be loaded may be optionally added to the system symbol table, depending on the value of *symFlag*:

##### LOAD\_NO\_SYMBOLS

add no symbols to the system symbol table

##### LOAD\_LOCAL\_SYMBOLS

add only local symbols to the system symbol table

##### LOAD\_GLOBAL\_SYMBOLS

add only external symbols to the system symbol table

##### LOAD\_ALL\_SYMBOLS

add both local and external symbols to the system symbol table

##### HIDDEN\_MODULE

do not display the module via *moduleShow()*.

In addition, the following symbols are also added to the symbol table to indicate the start of each segment: *filename\_text*, *filename\_data*, and *filename\_bss*, where *filename* is the name associated with the fd.

#### RELOCATION

The relocation commands in the object module are used to relocate the text, data, and BSS segments of the module. The location of each segment can be specified explicitly, or left unspecified in which case memory will be allocated for the segment from the system memory partition. This is determined by the parameters *ppText*, *ppData*, and *ppBss*, each of which can have the following values:

##### NULL

no load address is specified, none will be returned;

##### A pointer to LD\_NO\_ADDRESS

no load address is specified, the return address is referenced by the pointer;

##### A pointer to an address

the load address is specified.

The *ppText*, *ppData*, and *ppBss* parameters specify where to load the text, data, and bss sections respectively. Each of these parameters is a pointer to a pointer; for example, **\*\*ppText** gives the address where the text segment is to begin.

For any of the three parameters, there are two ways to request that new memory be allocated, rather than specifying the section's starting address: you can either specify the parameter itself as `NULL`, or you can write the constant `LD_NO_ADDRESS` in place of an address. In the second case, **loadModuleAt()** routine replaces the `LD_NO_ADDRESS` value with the address actually used for each section (that is, it records the address at *\*ppText*, *\*ppData*, or *\*ppBss*).

The double indirection not only permits reporting the addresses actually used, but also allows you to specify loading a segment at the beginning of memory, since the following cases can be distinguished:

- (1) Allocate memory for a section (text in this example): *ppText* == `NULL`
- (2) Begin a section at address zero (the text section, below): *\*ppText* == `0`

Note that **loadModule()** is equivalent to this routine if all three of the segment-address parameters are set to `NULL`.

**COMMON**

Some host compiler/linker combinations internally use another storage class known as *common*. In the C language, uninitialized global variables are eventually put in the BSS segment. However, in partially linked object modules, they are flagged internally as common and the static linker (host) resolves these and places them in BSS as a final step in creating a fully linked object module. However, the VxWorks loader is most often used to load partially linked object modules. When the VxWorks loader encounters a variable labeled as common, memory for the variable is allocated, with **malloc()**, and the variable is entered in the system symbol table (if specified) at that address. Note that most UNIX loaders have an option that forces resolution of the common storage while leaving the module relocatable (e.g., with typical BSD UNIX loaders, use options **-rd**).

**EXAMPLES**

Load a module into allocated memory, but do not return segment addresses:

```
module_id = loadModuleAt (fd, LOAD_GLOBAL_SYMBOLS, NULL, NULL, NULL);
```

Load a module into allocated memory, and return segment addresses:

```
pText = pData = pBss = LD_NO_ADDRESS;
module_id = loadModuleAt (fd, LOAD_GLOBAL_SYMBOLS, &pText, &pData,
&pBss);
```

Load a module to off-board memory at a specified address:

```
pText = 0x800000; /* address of text segment */
pData = pBss = LD_NO_ADDRESS /* other segments follow by default */
module_id = loadModuleAt (fd, LOAD_GLOBAL_SYMBOLS, &pText, &pData,
&pBss);
```

## *loattach()*

- RETURNS** `MODULE_ID`, or `NULL` if the file cannot be read, there is not enough memory, or the file format is illegal.
- SEE ALSO** `loadLib`, *VxWorks Programmer's Guide: Basic OS*

---

## *loattach()*

- NAME** *loattach()* – publish the **lo** network interface and initialize the driver and pseudo-device
- SYNOPSIS** `STATUS loattach (void)`
- DESCRIPTION** This routine attaches an **lo** Ethernet interface to the network, if the interface exists. It makes the interface available by filling in the network interface record. The system initializes the interface when it is ready to accept packets.
- RETURNS** `OK`.
- SEE ALSO** `if_loop`

---

## *localeconv()*

- NAME** *localeconv()* – set the components of an object with type **lconv** (ANSI)
- SYNOPSIS** `struct lconv *localeconv (void)`
- DESCRIPTION** This routine sets the components of an object with type **struct lconv** with values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.
- The members of the structure with type **char \*** are pointers to strings any of which (except **decimal\_point**) can point to "" to indicate that the value is not available in the current locale or is of zero length. The members with type **char** are nonnegative numbers, any of which can be `CHAR_MAX` to indicate that the value is not available in the current locale. The members include the following:
- char \*decimal\_point**  
The decimal-point character used to format nonmonetary quantities.
- char \*thousands\_sep**  
The character used to separate groups of digits before the decimal-point character in

formatted nonmonetary quantities.

**char \*grouping**

A string whose elements indicate the size of each group of digits in formatted nonmonetary quantities.

**char \*int\_curr\_symbol**

The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in ISO 4217:1987. The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

**char \*currency\_symbol**

The local currency symbol applicable to the current locale.

**char \*mon\_decimal\_point**

The decimal-point used to format monetary quantities.

**char \*mon\_thousands\_sep**

The separator for groups of digits before the decimal-point in formatted monetary quantities.

**char \*mon\_grouping**

A string whose elements indicate the size of each group of digits in formatted monetary quantities.

**char \*positive\_sign**

The string used to indicate a nonnegative-valued formatted monetary quantity.

**char \*negative\_sign**

The string used to indicate a negative-valued formatted monetary quantity.

**char int\_frac\_digits**

The number of fractional digits (those after the decimal-point) to be displayed in an internationally formatted monetary quantity.

**char frac\_digits**

The number of fractional digits (those after the decimal-point) to be displayed in a formatted monetary quantity.

**char p\_cs\_precedes**

Set to 1 or 0 if the **currency\_symbol** respectively precedes or succeeds the value for a nonnegative formatted monetary quantity.

**char p\_sep\_by\_space**

Set to 1 or 0 if the **currency\_symbol** respectively is or is not separated by a space from the value for a nonnegative formatted monetary quantity.

**char n\_cs\_precedes**

Set to 1 or 0 if the **currency\_symbol** respectively precedes or succeeds the value for a negative formatted monetary quantity.

***localeconv()***

**char n\_sep\_by\_space**

Set to 1 or 0 if the **currency\_symbol** respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

**char p\_sign\_posn**

Set to a value indicating the positioning of the **positive\_sign** for a nonnegative formatted monetary quantity.

**char n\_sign\_posn**

Set to a value indicating the positioning of the **negative\_sign** for a negative formatted monetary quantity.

The elements of **grouping** and **mon\_grouping** are interpreted according to the following:

**CHAR\_MAX**

No further grouping is to be performed.

0

The previous element is to be repeatedly used for the remainder of the digits.

other

The integer value is the number of the digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The values of **p\_sign\_posn** and **n\_sign\_posn** are interpreted according to the following:

0 Parentheses surround the quantity and **currency\_symbol**.

1 The sign string precedes the quantity and **currency\_symbol**.

2 The sign string succeeds the quantity and **currency\_symbol**.

3 The sign string immediately precedes the **currency\_symbol**.

4 The sign string immediately succeeds the **currency\_symbol**.

The implementation behaves as if no library function calls **localeconv()**.

The **localeconv()** routine returns a pointer to the filled-in object. The structure pointed to by the return value is not modified by the program, but may be overwritten by a subsequent call to **localeconv()**. In addition, calls to **setlocale()** with categories **LC\_ALL**, **LC\_MONETARY**, or **LC\_NUMERIC** may overwrite the contents of the structure.

**INCLUDE FILES**     **locale.h, limits.h**

**RETURNS**            A pointer to the structure **lconv**.

**SEE ALSO**            **ansiLocale**



---

## *localtime()*

|                      |                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>localtime()</i> – convert calendar time into broken-down time (ANSI)                                                                                                                                            |
| <b>SYNOPSIS</b>      | <pre>struct tm *localtime (     const time_t * timer /* calendar time in seconds */ )</pre>                                                                                                                        |
| <b>DESCRIPTION</b>   | <p>This routine converts the calendar time pointed to by <i>timer</i> into broken-down time, expressed as local time.</p> <p>This routine is not reentrant. For a reentrant version, see <i>localtime_r()</i>.</p> |
| <b>INCLUDE FILES</b> | <code>time.h</code>                                                                                                                                                                                                |
| <b>RETURNS</b>       | A pointer to a <code>tm</code> structure containing the local broken-down time.                                                                                                                                    |
| <b>SEE ALSO</b>      | <code>ansiTime</code>                                                                                                                                                                                              |

---

## *localtime\_r()*

|                      |                                                                                                                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>localtime_r()</i> – convert calendar time into broken-down time (POSIX)                                                                                                                                                                                |
| <b>SYNOPSIS</b>      | <pre>int localtime_r (     const time_t * timer,      /* calendar time in seconds */     struct tm *   timeBuffer /* buffer for the broken-down time */ )</pre>                                                                                           |
| <b>DESCRIPTION</b>   | <p>This routine converts the calendar time pointed to by <i>timer</i> into broken-down time, expressed as local time. The broken-down time is stored in <i>timeBuffer</i>.</p> <p>This routine is the POSIX re-entrant version of <i>localtime()</i>.</p> |
| <b>INCLUDE FILES</b> | <code>time.h</code>                                                                                                                                                                                                                                       |
| <b>RETURNS</b>       | OK.                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>      | <code>ansiTime</code>                                                                                                                                                                                                                                     |

**log()**

---

**log()**

|                      |                                                                                                                                                                                                                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>log()</i> – compute a natural logarithm (ANSI)                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>      | <pre>double log (     double x /* value to compute the natural logarithm of */ )</pre>                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b>   | This routine returns the natural logarithm of <i>x</i> in double precision (IEEE double, 53 bits). A domain error occurs if the argument is negative. A range error may occur if the argument is zero.                                                                                                          |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                                                                                                                                                                                   |
| <b>RETURNS</b>       | The double-precision natural logarithm of <i>x</i> .<br>Special cases:<br>If <i>x</i> < 0 (including -INF), it returns NaN with signal.<br>If <i>x</i> is +INF, it returns <i>x</i> with no signal.<br>If <i>x</i> is 0, it returns -INF with signal.<br>If <i>x</i> is NaN it returns <i>x</i> with no signal. |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                                                                                                                                                                                                                                       |

---

**log2()**

|                      |                                                                                          |
|----------------------|------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>log2()</i> – compute a base-2 logarithm                                               |
| <b>SYNOPSIS</b>      | <pre>double log2 (     double x /* value to compute the base-two logarithm of */ )</pre> |
| <b>DESCRIPTION</b>   | This routine returns the base-2 logarithm of <i>x</i> in double precision.               |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                            |
| <b>RETURNS</b>       | The double-precision base-2 logarithm of <i>x</i> .                                      |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                          |

---

## *log2f()*

|                      |                                                                                       |
|----------------------|---------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>log2f()</i> – compute a base-2 logarithm                                           |
| <b>SYNOPSIS</b>      | <pre>float log2f (     float x /* value to compute the base-2 logarithm of */ )</pre> |
| <b>DESCRIPTION</b>   | This routine returns the base-2 logarithm of <i>x</i> in single precision.            |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                         |
| <b>RETURNS</b>       | The single-precision base-2 logarithm of <i>x</i> .                                   |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                       |

---

## *log10()*

|                      |                                                                                                                                                                                                                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>log10()</i> – compute a base-10 logarithm (ANSI)                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>      | <pre>double log10 (     double x /* value to compute the base-10 logarithm of */ )</pre>                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>   | This routine returns the base 10 logarithm of <i>x</i> in double precision (IEEE double, 53 bits).<br>A domain error occurs if the argument is negative. A range error may if the argument is zero.                                                                                                        |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                                                                                                                                                                              |
| <b>RETURNS</b>       | The double-precision base-10 logarithm of <i>x</i> .<br>Special cases:<br>If <i>x</i> < 0, <i>log10()</i> returns NaN with signal.<br>if <i>x</i> is +INF, it returns <i>x</i> with no signal.<br>if <i>x</i> is 0, it returns -INF with signal.<br>if <i>x</i> is NaN it returns <i>x</i> with no signal. |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                                                                                                                                                                                                                                  |

## *log10f()*

**NAME** *log10f()* – compute a base-10 logarithm (ANSI)

**SYNOPSIS**

```
float log10f
(
 float x /* value to compute the base-10 logarithm of */
)
```

**DESCRIPTION** This routine returns the base-10 logarithm of  $x$  in single precision.

**INCLUDE FILES** `math.h`

**RETURNS** The single-precision base-10 logarithm of  $x$ .

**SEE ALSO** `mathALib`

---

## *logf()*

**NAME** *logf()* – compute a natural logarithm (ANSI)

**SYNOPSIS**

```
float logf
(
 float x /* value to compute the natural logarithm of */
)
```

**DESCRIPTION** This routine returns the logarithm of  $x$  in single precision.

**INCLUDE FILES** `math.h`

**RETURNS** The single-precision natural logarithm of  $x$ .

**SEE ALSO** `mathALib`

---

## *logFdAdd()*

|                    |                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>logFdAdd()</i> – add a logging file descriptor                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>STATUS logFdAdd (     int fd /* file descriptor for additional logging device */ )</pre>                                                                                   |
| <b>DESCRIPTION</b> | This routine adds to the log file descriptor list another file descriptor <i>fd</i> to which messages will be logged. The file descriptor must be a valid open file descriptor. |
| <b>RETURNS</b>     | OK, or ERROR if the allowable number of additional logging file descriptors (5) is exceeded.                                                                                    |
| <b>SEE ALSO</b>    | <i>logLib</i> , <i>logFdDelete()</i>                                                                                                                                            |

---

## *logFdDelete()*

|                    |                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>logFdDelete()</i> – delete a logging file descriptor                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>STATUS logFdDelete (     int fd /* file descriptor to stop using as logging device */ )</pre>                                                                                                |
| <b>DESCRIPTION</b> | This routine removes from the log file descriptor list a logging file descriptor added by <i>logFdAdd()</i> . The file descriptor is not closed; but is no longer used by the logging facilities. |
| <b>RETURNS</b>     | OK, or ERROR if the file descriptor was not added with <i>logFdAdd()</i> .                                                                                                                        |
| <b>SEE ALSO</b>    | <i>logLib</i> , <i>logFdAdd()</i>                                                                                                                                                                 |

**logFdSet()**

---

**logFdSet()**

**NAME** *logFdSet()* – set the primary logging file descriptor

**SYNOPSIS**

```
void logFdSet
(
 int fd /* file descriptor to use as logging device */
)
```

**DESCRIPTION** This routine changes the file descriptor where messages from *logMsg()* are written, allowing the log device to be changed from the default specified by *logInit()*. It first removes the old file descriptor (if one had been previously set) from the log file descriptor list, then adds the new *fd*.

The old logging file descriptor is not closed or affected by this call; it is simply no longer used by the logging facilities.

**RETURNS** N/A

**SEE ALSO** *logLib*, *logFdAdd()*, *logFdDelete()*

---

**loginDefaultEncrypt()**

**NAME** *loginDefaultEncrypt()* – default password encryption routine

**SYNOPSIS**

```
STATUS loginDefaultEncrypt
(
 char * in, /* input string */
 char * out /* encrypted string */
)
```

**DESCRIPTION** This routine provides default encryption for login passwords. It employs a simple encryption algorithm. It takes as arguments a string *in* and a pointer to a buffer *out*. The encrypted string is then stored in the buffer.

The input strings must be at least 8 characters and no more than 40 characters.

If a more sophisticated encryption algorithm is needed, this routine can be replaced, as long as the new encryption routine retains the same declarations as the default routine. The routine *vxencrypt* in *host/hostOs/bin* should also be replaced by a host version of *encryptionRoutine*. For more information, see the manual entry for *loginEncryptInstall()*.

**RETURNS** OK, or ERROR if the password is invalid.

**SEE ALSO** loginLib, loginEncryptInstall(), vxencrypt

---

## loginEncryptInstall()

**NAME** loginEncryptInstall() – install an encryption routine

**SYNOPSIS**

```
void loginEncryptInstall
(
 FUNCPTR rtn, /* function pointer to encryption routine */
 int var /* argument to the encryption routine (unused) */
)
```

**DESCRIPTION** This routine allows the user to install a custom encryption routine. The custom routine *rtn* must be of the following form:

```
STATUS encryptRoutine
(
 char *password, /* string to encrypt */
 char *encryptedPassword /* resulting encryption */
)
```

When a custom encryption routine is installed, a host version of this routine must be written to replace the tool vxencrypt in **host/hostOs/bin**.

**EXAMPLE** The custom example above could be installed as follows:

```
#ifdef INCLUDE_SECURITY
 loginInit (); /* initialize login table */
 shellLoginInstall (loginPrompt, NULL); /* install shell security */
 loginEncryptInstall (encryptRoutine, NULL); /* install encrypt routine */
#endif
```

**RETURNS** N/A

**SEE ALSO** loginLib, loginDefaultEncrypt(), vxencrypt

## *loginInit()*

|                    |                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>loginInit()</i> – initialize the login table                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <code>void loginInit (void)</code>                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | This routine must be called to initialize the login data structure used by routines throughout this module. If the configuration macro <b>INCLUDE_SECURITY</b> is defined, it is called by <i>usrRoot()</i> in <b>usrConfig.c</b> , before any other routines in this module. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>loginLib</b>                                                                                                                                                                                                                                                               |

---

## *logInit()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>logInit()</i> – initialize message logging library                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>STATUS logInit (     int fd,      /* file descriptor to use as logging device */     int maxMsgs /* max. number of messages allowed in log queue */ )</pre>                                                                                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This routine specifies the file descriptor to be used as the logging device and the number of messages that can be in the logging queue. If more than <i>maxMsgs</i> are in the queue, they will be discarded. A message is printed to indicate lost messages.</p> <p>This routine spawns <i>logTask()</i>, the task-level portion of error logging.</p> <p>This routine must be called before any other routine in <b>logLib</b>. This is done by the root task, <i>usrRoot()</i>, in <b>usrConfig.c</b>.</p> |
| <b>RETURNS</b>     | OK, or ERROR if a message queue could not be created or <i>logTask()</i> could not be spawned.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <b>logLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |



---

## *loginPrompt()*

|                    |                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>loginPrompt()</i> – display a login prompt and validate a user entry                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>STATUS loginPrompt (     char * userName /* user name, ask if NULL or not provided */ )</pre>                                                                                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | <p>This routine displays a login prompt and validates a user entry. If both user name and password match with an entry in the login table, the user is then given access to the VxWorks system. Otherwise, it prompts the user again.</p> <p>All control characters are disabled during authentication except CTRL-D, which will terminate the remote login session.</p> |
| <b>RETURNS</b>     | OK if the name and password are valid, or ERROR if there is an EOF or the routine times out.                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <b>loginLib</b>                                                                                                                                                                                                                                                                                                                                                          |

---

## *loginStringSet()*

|                    |                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>loginStringSet()</i> – change the login string                                                                    |
| <b>SYNOPSIS</b>    | <pre>void loginStringSet (     char * newString /* string to become new login prompt */ )</pre>                      |
| <b>DESCRIPTION</b> | <p>This routine changes the login prompt string to <i>newString</i>. The maximum string length is 80 characters.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                  |
| <b>SEE ALSO</b>    | <b>loginLib</b>                                                                                                      |

---

## *loginUserAdd()*

**NAME** *loginUserAdd()* – add a user to the login table

**SYNOPSIS**

```
STATUS loginUserAdd
(
 char name[MAX_LOGIN_NAME_LEN+1], /* user name */
 char passwd[80] /* user password */
)
```

**DESCRIPTION** This routine adds a user name and password entry to the login table. Note that what is saved in the login table is the user name and the address of *passwd*, not the actual password.

The length of user names should not exceed `MAX_LOGIN_NAME_LEN`, while the length of passwords depends on the encryption routine used. For the default encryption routine, passwords should be at least 8 characters long and no more than 40 characters.

The procedure for adding a new user to login table is as follows:

- (1) Generate the encrypted password by invoking `vxencrypt` in `host/hostOs/bin`.
- (2) Add a user by invoking `loginUserAdd()` in the VxWorks shell with the user name and the encrypted password.

The password of a user can be changed by first deleting the user entry, then adding the user entry again with the new encrypted password.

**EXAMPLE**

```
-> loginUserAdd "peter", "RRdRd9Qbyz"
value = 0 = 0x0
-> loginUserAdd "robin", "bSzydqbsb"
value = 0 = 0x0
-> loginUserShow
 User Name
 =====
 peter
 robin
value = 0 = 0x0
->
```

**RETURNS** OK, or ERROR if the user name has already been entered.

**SEE ALSO** `loginLib`, `vxencrypt`

---

## *loginUserDelete()*

**NAME** *loginUserDelete()* – delete a user entry from the login table

**SYNOPSIS** `STATUS loginUserDelete`  
`(`  
`char * name, /* user name */`  
`char * passwd /* user password */`  
`)`

**DESCRIPTION** This routine deletes an entry in the login table. Both the user name and password must be specified to remove an entry from the login table.

**RETURNS** OK, or ERROR if the specified user or password is incorrect.

**SEE ALSO** **loginLib**

---

## *loginUserShow()*

**NAME** *loginUserShow()* – display the user login table

**SYNOPSIS** `void loginUserShow (void)`

**DESCRIPTION** This routine displays valid user names.

**EXAMPLE**

```
-> loginUserShow ()
 User Name
 =====
 peter
 robin
 value = 0 = 0x0
```

**RETURNS** N/A

**SEE ALSO** **loginLib**

## *loginUserVerify()*

|                    |                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>loginUserVerify()</i> – verify a user name and password in the login table                                      |
| <b>SYNOPSIS</b>    | <pre>STATUS loginUserVerify (     char * name, /* name of user */     char * passwd /* password of user */ )</pre> |
| <b>DESCRIPTION</b> | This routine verifies a user entry in the login table.                                                             |
| <b>RETURNS</b>     | OK, or ERROR if the user name or password is not found.                                                            |
| <b>SEE ALSO</b>    | <i>loginLib</i>                                                                                                    |

---

## *logMsg()*

|                               |                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>                   | <i>logMsg()</i> – log a formatted error message                                                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>               | <pre>int logMsg (     char * fmt, /* format string for print */     int  arg1, /* first of six required args for fmt */     int  arg2,     int  arg3,     int  arg4,     int  arg5,     int  arg6 )</pre>                                                                                                                            |
| <b>DESCRIPTION</b>            | <p>This routine logs a specified message via the logging task. This routine's syntax is similar to <i>printf()</i> -- a format string is followed by arguments to format. However, the <i>logMsg()</i> routine requires a fixed number of arguments (6).</p> <p>The task ID of the caller is prepended to the specified message.</p> |
| <b>SPECIAL CONSIDERATIONS</b> | <p>Because <i>logMsg()</i> does not actually perform the output directly to the logging streams, but instead queues the message to the logging task, <i>logMsg()</i> can be called from interrupt service routines.</p>                                                                                                              |

However, since the arguments are interpreted by the *logTask()* at the time of actual logging, instead of at the moment when *logMsg()* is called, arguments to *logMsg()* should not be pointers to volatile entities (e.g., dynamic strings on the caller stack).

For more detailed information about the use of *logMsg()*, see the manual entry for **logLib**.

**EXAMPLE**

If the following code were executed by task 20:

```
{
 name = "GRONK";
 num = 123;
 logMsg ("ERROR - name = %s, num = %d.\n", name, num, 0, 0, 0, 0);
}
```

the following error message would appear on the system log:

```
0x180400 (t20): ERROR - name = GRONK, num = 123.
```

**RETURNS**

The number of bytes written to the log queue, or EOF if the routine is unable to write a message.

**SEE ALSO**

**logLib**, *printf()*, *logTask()*

---

## *logout()*

**NAME**

*logout()* – log out of the VxWorks system

**SYNOPSIS**

```
void logout (void)
```

**DESCRIPTION**

This command logs out of the VxWorks shell. If a remote login is active (via **rlogin** or **telnet**), it is stopped, and standard I/O is restored to the console.

**SEE ALSO**

**usrLib**, *rlogin()*, *telnet()*, *shellLogout()*, *VxWorks Programmer's Guide: Target Shell*

---

## *logTask()*

**NAME**

*logTask()* – message-logging support task

**SYNOPSIS**

```
void logTask (void)
```

**longjmp()**

|                    |                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine prints the messages logged with <i>logMsg()</i> . It waits on a message queue and prints the messages as they arrive on the file descriptor specified by <i>logInit()</i> (or a subsequent call to <i>logFdSet()</i> or <i>logFdAdd()</i> ).<br><br>This task is spawned by <i>logInit()</i> . |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <i>logLib</i> , <i>logMsg()</i>                                                                                                                                                                                                                                                                             |

---

## *longjmp()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>longjmp()</i> – perform non-local goto by restoring saved environment (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>void longjmp (     jmp_buf env,     int     val )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>DESCRIPTION</b> | <p>This routine restores the environment saved by the most recent invocation of the <i>setjmp()</i> routine that used the same <i>jmp_buf</i> specified in the argument <i>env</i>. The restored environment includes the program counter, thus transferring control to the <i>setjmp()</i> caller.</p> <p>If there was no corresponding <i>setjmp()</i> call, or if the function containing the corresponding <i>setjmp()</i> routine call has already returned, the behavior of <i>longjmp()</i> is unpredictable.</p> <p>All accessible objects in memory retain their values as of the time <i>longjmp()</i> was called, with one exception: local objects on the C stack that are not declared <b>volatile</b>, and have been changed between the <i>setjmp()</i> invocation and the <i>longjmp()</i> call, have unpredictable values.</p> <p>The <i>longjmp()</i> function executes correctly in contexts of signal handlers and any of their associated functions (but not from interrupt handlers).</p> |
| <b>WARNING</b>     | Do not use <i>longjmp()</i> or <i>setjmp()</i> from an ISR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>RETURNS</b>     | This routine does not return to its caller. Instead, it causes <i>setjmp()</i> to return <i>val</i> , unless <i>val</i> is 0; in that case <i>setjmp()</i> returns 1.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>ansiSetjmp</i> , <i>setjmp()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

---

## *lptDevCreate()*

|                    |                                                                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lptDevCreate()</i> – create a device for an LPT port                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>STATUS lptDevCreate (     char * name,    /* name to use for this device */     int   channel /* physical channel for this device (0 - 2) */ )</pre>                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine creates a device for a specified LPT port. Each port to be used should have exactly one device associated with it by calling this routine.</p> <p>For instance, to create the device <i>/lpt/0</i>, the proper call would be:</p> <pre>lptDevCreate ("/lpt/0", 0);</pre> |
| <b>RETURNS</b>     | OK, or ERROR if the driver is not installed, the channel is invalid, or the device already exists.                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <i>lptDrv()</i>                                                                                                                                                                                                                                                                          |

---

## *lptDrv()*

|                    |                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lptDrv()</i> – initialize the LPT driver                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>STATUS lptDrv (     int           channels, /* LPT channels */     LPT_RESOURCE * pResource /* LPT resources */ )</pre>                                                                                                                                                                                        |
| <b>DESCRIPTION</b> | <p>This routine initializes the LPT driver, sets up interrupt vectors, and performs hardware initialization of the LPT ports.</p> <p>This routine should be called exactly once, before any reads, writes, or calls to <i>lptDevCreate()</i>. Normally, it is called by <i>usrRoot()</i> in <i>usrConfig.c</i>.</p> |
| <b>RETURNS</b>     | OK, or ERROR if the driver cannot be installed.                                                                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <i>lptDrv</i> , <i>lptDevCreate()</i>                                                                                                                                                                                                                                                                               |

***lptShow()***

---

***lptShow()***

|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <b>NAME</b>        | <i>lptShow()</i> – show LPT statistics                             |
| <b>SYNOPSIS</b>    | <pre>void lptShow (     UINT channel /* channel (0 - 2) */ )</pre> |
| <b>DESCRIPTION</b> | This routine shows statistics for a specified LPT port.            |
| <b>RETURNS</b>     | N/A                                                                |
| <b>SEE ALSO</b>    | <b>lptDrv</b>                                                      |

---

***ls()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ls()</i> – list the contents of a directory                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>STATUS ls (     char * dirName, /* name of dir to list */     BOOL doLong /* if TRUE, do long listing */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b> | <p>This command is similar to UNIX <i>ls</i>. It lists the contents of a directory in one of two formats. If <i>doLong</i> is FALSE, only the names of the files (or subdirectories) in the specified directory are displayed. If <i>doLong</i> is TRUE, then the file name, size, date, and time are displayed. For a long listing, any entries that describe subdirectories are also flagged with the label "DIR".</p> <p>The <i>dirName</i> parameter specifies which directory to list. If <i>dirName</i> is omitted or NULL, the current working directory is listed.</p> <p>Empty directory entries and dosFs volume label entries are not reported.</p> |
| <b>NOTE</b>        | When used with <b>netDrv</b> devices (FTP or RSH), <i>doLong</i> has no effect.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <b>usrLib, ll(), lsOld(), stat(), VxWorks Programmer's Guide: Target Shell, windsh, Tornado User's Guide: Shell</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |



---

## *lseek()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |              |                        |              |                                              |              |                                                  |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|------------------------|--------------|----------------------------------------------|--------------|--------------------------------------------------|
| <b>NAME</b>        | <i>lseek()</i> – set a file read/write pointer                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |              |                        |              |                                              |              |                                                  |
| <b>SYNOPSIS</b>    | <pre>int lseek (     int fd,      /* file descriptor */     long offset, /* new byte offset to seek to */     int whence  /* relative file position */ )</pre>                                                                                                                                                                                                                                                                                                                                                                  |              |                        |              |                                              |              |                                                  |
| <b>DESCRIPTION</b> | <p>This routine sets the file read/write pointer of file <i>fd</i> to <i>offset</i>. The argument <i>whence</i>, which affects the file position pointer, has three values:</p> <table><tr><td>SEEK_SET (0)</td><td>- set to <i>offset</i></td></tr><tr><td>SEEK_CUR (1)</td><td>- set to current position plus <i>offset</i></td></tr><tr><td>SEEK_END (2)</td><td>- set to the size of the file plus <i>offset</i></td></tr></table> <p>This routine calls <i>ioctl()</i> with functions FIOWHERE, FIONREAD, and FIOSEEK.</p> | SEEK_SET (0) | - set to <i>offset</i> | SEEK_CUR (1) | - set to current position plus <i>offset</i> | SEEK_END (2) | - set to the size of the file plus <i>offset</i> |
| SEEK_SET (0)       | - set to <i>offset</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |              |                        |              |                                              |              |                                                  |
| SEEK_CUR (1)       | - set to current position plus <i>offset</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |              |                        |              |                                              |              |                                                  |
| SEEK_END (2)       | - set to the size of the file plus <i>offset</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |              |                        |              |                                              |              |                                                  |
| <b>RETURNS</b>     | The new offset from the beginning of the file, or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |              |                        |              |                                              |              |                                                  |
| <b>SEE ALSO</b>    | <i>ioLib</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |              |                        |              |                                              |              |                                                  |

---

## *lsOld()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lsOld()</i> – list the contents of an RT-11 directory                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>STATUS lsOld (     char * dirName /* device to list */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | <p>This command is the old version of <i>ls()</i>, which used the old-style <i>ioctl()</i> function FIODIRENTRY to get information about entries in a directory. Since VxWorks 5.0, a new version of <i>ls()</i>, which uses POSIX directory and file functions, has replaced the older routine.</p> <p>This version remains in the system to support certain drivers that do not currently support the POSIX directory and file functions. This includes <b>netDrv</b>, which provides the Remote Shell (RSH) and File Transfer Protocol (FTP) mode remote file access (although</p> |

## ***lstAdd()***

**nfsDrv**, which uses NFS, does support the directory calls). Also, the new **ls()** no longer reports empty directory entries on RT-11 disks (i.e., the entries that describe unallocated sections of an RT-11 disk).

If no directory name is specified, the current working directory is listed.

**RETURNS** OK, or ERROR if the directory cannot be opened.

**SEE ALSO** **usrLib**, **ls()**, *VxWorks Programmer's Guide: Target Shell*

---

## ***lstAdd()***

**NAME** *lstAdd()* – add a node to the end of a list

**SYNOPSIS**

```
void lstAdd
(
 LIST * pList, /* pointer to list descriptor */
 NODE * pNode /* pointer to node to be added */
)
```

**DESCRIPTION** This routine adds a specified node to the end of a specified list.

**RETURNS** N/A

**SEE ALSO** **lstLib**

---

## ***lstConcat()***

**NAME** *lstConcat()* – concatenate two lists

**SYNOPSIS**

```
void lstConcat
(
 LIST * pDstList, /* destination list */
 LIST * pAddList /* list to be added to dstList */
)
```

**DESCRIPTION** This routine concatenates the second list to the end of the first list. The second list is left empty. Either list (or both) can be empty at the beginning of the operation.

**RETURNS** N/A

**SEE ALSO** `lstLib`

---

## *lstCount()*

**NAME** *lstCount()* – report the number of nodes in a list

**SYNOPSIS**

```
int lstCount
(
 LIST * pList /* pointer to list descriptor */
)
```

**DESCRIPTION** This routine returns the number of nodes in a specified list.

**RETURNS** The number of nodes in the list.

**SEE ALSO** `lstLib`

---

## *lstDelete()*

**NAME** *lstDelete()* – delete a specified node from a list

**SYNOPSIS**

```
void lstDelete
(
 LIST * pList, /* pointer to list descriptor */
 NODE * pNode /* pointer to node to be deleted */
)
```

**DESCRIPTION** This routine deletes a specified node from a specified list.

**RETURNS** N/A

**SEE ALSO** `lstLib`

## ***lstExtract()***

**NAME** *lstExtract()* – extract a sublist from a list

**SYNOPSIS**

```
void lstExtract
(
 LIST * pSrcList, /* pointer to source list */
 NODE * pStartNode, /* first node in sublist to be extracted */
 NODE * pEndNode, /* last node in sublist to be extracted */
 LIST * pDstList /* ptr to list where to put extracted list */
)
```

**DESCRIPTION** This routine extracts the sublist that starts with *pStartNode* and ends with *pEndNode* from a source list. It places the extracted list in *pDstList*.

**RETURNS** N/A

**SEE ALSO** *lstLib*

---

## ***lstFind()***

**NAME** *lstFind()* – find a node in a list

**SYNOPSIS**

```
int lstFind
(
 LIST * pList, /* list in which to search */
 NODE * pNode /* pointer to node to search for */
)
```

**DESCRIPTION** This routine returns the node number of a specified node (the first node is 1).

**RETURNS** The node number, or ERROR if the node is not found.

**SEE ALSO** *lstLib*

---

## *lstFirst()*

**NAME** *lstFirst()* – find first node in list

**SYNOPSIS**

```
NODE *lstFirst
(
 LIST * pList /* pointer to list descriptor */
)
```

**DESCRIPTION** This routine finds the first node in a linked list.

**RETURNS** A pointer to the first node in a list, or NULL if the list is empty.

**SEE ALSO** *lstLib*

---

## *lstFree()*

**NAME** *lstFree()* – free up a list

**SYNOPSIS**

```
void lstFree
(
 LIST * pList /* list for which to free all nodes */
)
```

**DESCRIPTION** This routine turns any list into an empty list. It also frees up memory used for nodes.

**RETURNS** N/A

**SEE ALSO** *lstLib, free()*

***lstGet()***

---

***lstGet()***

**NAME** *lstGet()* – delete and return the first node from a list

**SYNOPSIS**

```
NODE *lstGet
(
 LIST * pList /* ptr to list from which to get node */
)
```

**DESCRIPTION** This routine gets the first node from a specified list, deletes the node from the list, and returns a pointer to the node gotten.

**RETURNS** A pointer to the node gotten, or NULL if the list is empty.

**SEE ALSO** *lstLib*

---

***lstInit()***

**NAME** *lstInit()* – initialize a list descriptor

**SYNOPSIS**

```
void lstInit
(
 LIST * pList /* ptr to list descriptor to be initialized */
)
```

**DESCRIPTION** This routine initializes a specified list to an empty list.

**RETURNS** N/A

**SEE ALSO** *lstLib*

---

## *lstInsert()*

|                    |                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lstInsert()</i> – insert a node in a list after a specified node                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>void lstInsert (     LIST * pList, /* pointer to list descriptor */     NODE * pPrev, /* pointer to node after which to insert */     NODE * pNode /* pointer to node to be inserted */ )</pre> |
| <b>DESCRIPTION</b> | This routine inserts a specified node in a specified list. The new node is placed following the list node <i>pPrev</i> . If <i>pPrev</i> is NULL, the node is inserted at the head of the list.      |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <i>lstLib</i>                                                                                                                                                                                        |

---

## *lstLast()*

|                    |                                                                                |
|--------------------|--------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lstLast()</i> – find the last node in a list                                |
| <b>SYNOPSIS</b>    | <pre>NODE *lstLast (     LIST * pList /* pointer to list descriptor */ )</pre> |
| <b>DESCRIPTION</b> | This routine finds the last node in a list.                                    |
| <b>RETURNS</b>     | A pointer to the last node in the list, or NULL if the list is empty.          |
| <b>SEE ALSO</b>    | <i>lstLib</i>                                                                  |

***lstNext()***

---

***lstNext()***

|                    |                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lstNext()</i> – find the next node in a list                                                  |
| <b>SYNOPSIS</b>    | <pre> NODE *lstNext (     NODE * pNode /* ptr to node whose successor is to be found */ ) </pre> |
| <b>DESCRIPTION</b> | This routine locates the node immediately following a specified node.                            |
| <b>RETURNS</b>     | A pointer to the next node in the list, or NULL if there is no next node.                        |
| <b>SEE ALSO</b>    | lstLib                                                                                           |

---

***lstNStep()***

|                    |                                                                                                                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lstNStep()</i> – find a list node <i>nStep</i> steps away from a specified node                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre> NODE *lstNStep (     NODE * pNode, /* the known node */     int   nStep /* number of steps away to find */ ) </pre>                                                                                                                                                |
| <b>DESCRIPTION</b> | This routine locates the node <i>nStep</i> steps away in either direction from a specified node. If <i>nStep</i> is positive, it steps toward the tail. If <i>nStep</i> is negative, it steps toward the head. If the number of steps is out of range, NULL is returned. |
| <b>RETURNS</b>     | A pointer to the node <i>nStep</i> steps away, or NULL if the node is out of range.                                                                                                                                                                                      |
| <b>SEE ALSO</b>    | lstLib                                                                                                                                                                                                                                                                   |



---

## *lstNth()*

|                    |                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lstNth()</i> – find the Nth node in a list                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>NODE *lstNth (     LIST * pList, /* pointer to list descriptor */     int   nodenum /* number of node to be found */ )</pre>                                                                                                                                                                         |
| <b>DESCRIPTION</b> | This routine returns a pointer to the node specified by a number <i>nodenum</i> where the first node in the list is numbered 1. Note that the search is optimized by searching forward from the beginning if the node is closer to the head, and searching back from the end if it is closer to the tail. |
| <b>RETURNS</b>     | A pointer to the Nth node, or NULL if there is no Nth node.                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <i>lstLib</i>                                                                                                                                                                                                                                                                                             |

---

## *lstPrevious()*

|                    |                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>lstPrevious()</i> – find the previous node in a list                                              |
| <b>SYNOPSIS</b>    | <pre>NODE *lstPrevious (     NODE * pNode /* ptr to node whose predecessor is to be found */ )</pre> |
| <b>DESCRIPTION</b> | This routine locates the node immediately preceding the node pointed to by <i>pNode</i> .            |
| <b>RETURNS</b>     | A pointer to the previous node in the list, or NULL if there is no previous node.                    |
| <b>SEE ALSO</b>    | <i>lstLib</i>                                                                                        |

***m()***

---

***m()*****NAME** *m()* – modify memory**SYNOPSIS**  

```
void m
(
 void * adrs, /* address to change */
 int width /* width of unit to be modified (1, 2, 4, 8) */
)
```

**DESCRIPTION** This command prompts the user for modifications to memory in byte, short word, or long word specified by *width*, starting at the specified address. It prints each address and the current contents of that address, in turn. If *adrs* or *width* is zero or absent, it defaults to the previous value. The user can respond in one of several ways:**RETURN**

Do not change this address, but continue, prompting at the next address.

*number*Set the content of this address to *number*.

## . (dot)

Do not change this address, and quit.

## EOF

Do not change this address, and quit.

All numbers entered and displayed are in hexadecimal.

**RETURNS** N/A**SEE ALSO** *usrLib*, *mRegs()*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

---

***m2Delete()*****NAME** *m2Delete()* – delete all the MIB-II library groups**SYNOPSIS** `STATUS m2Delete (void)`**DESCRIPTION** This routine cleans up the state associated with the MIB-II library.**RETURNS** OK (always).

**SEE ALSO** *m2Lib*, *m2SysDelete()*, *m2TcpDelete()*, *m2UdpDelete()*, *m2IcmpDelete()*, *m2IfDelete()*, *m2IpDelete()*

---

## *m2IcmpDelete()*

**NAME** *m2IcmpDelete()* – delete all resources used to access the ICMP group

**SYNOPSIS** `STATUS m2IcmpDelete (void)`

**DESCRIPTION** This routine frees all the resources allocated at the time the ICMP group was initialized. The ICMP group should not be accessed after this routine has been called.

**RETURNS** OK, always.

**SEE ALSO** *m2IcmpLib*, *m2IcmpInit()*, *m2IcmpGroupInfoGet()*

---

## *m2IcmpGroupInfoGet()*

**NAME** *m2IcmpGroupInfoGet()* – get the MIB-II ICMP-group global variables

**SYNOPSIS** `STATUS m2IcmpGroupInfoGet  
(  
    M2_ICMP * pIcmpInfo /* pointer to the ICMP group structure */  
)`

**DESCRIPTION** This routine fills in the ICMP structure at *pIcmpInfo* with the MIB-II ICMP scalar variables.

**RETURNS** OK, or ERROR if the input parameter *pIcmpInfo* is invalid.

**ERRNO** `S_m2Lib_INVALID_PARAMETER`

**SEE ALSO** *m2IcmpLib*, *m2IcmpInit()*, *m2IcmpDelete()*

***m2IcmpInit()***

---

***m2IcmpInit()***

**NAME** *m2IcmpInit()* – initialize MIB-II ICMP-group access

**SYNOPSIS** `STATUS m2IcmpInit (void)`

**DESCRIPTION** This routine allocates the resources needed to allow access to the MIB-II ICMP-group variables. This routine must be called before any ICMP variables can be accessed.

**RETURNS** OK, always.

**SEE ALSO** *m2IcmpLib*, *m2IcmpGroupInfoGet()*, *m2IcmpDelete()*

---

***m2IfDelete()***

**NAME** *m2IfDelete()* – delete all resources used to access the interface group

**SYNOPSIS** `STATUS m2IfDelete (void)`

**DESCRIPTION** This routine frees all the resources allocated at the time the group was initialized. The interface group should not be accessed after this routine has been called.

**RETURNS** OK, always.

**SEE ALSO** *m2IfLib*, *m2IfInit()*, *m2IfGroupInfoGet()*, *m2IfTblEntryGet()*, *m2IfTblEntrySet()*

---

***m2IfGroupInfoGet()***

**NAME** *m2IfGroupInfoGet()* – get the MIB-II interface-group scalar variables

**SYNOPSIS** `STATUS m2IfGroupInfoGet  
(  
    M2_INTERFACE * pIfInfo /* pointer to interface group structure */  
)`

**DESCRIPTION** This routine fills out the interface-group structure at *pIfInfo* with the values of MIB-II interface-group global variables.

**RETURNS** OK, or ERROR if *pIfInfo* is not a valid pointer.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER

**SEE ALSO** m2IfLib, m2IfInit(), m2IfTblEntryGet(), m2IfTblEntrySet(), m2IfDelete()

---

## **m2IfInit()**

**NAME** m2IfInit() – initialize MIB-II interface-group routines

**SYNOPSIS**

```
STATUS m2IfInit
(
 FUNCPTR pTrapRtn, /* pointer to user trap generator */
 void * pTrapArg /* pointer to user trap generator argument */
)
```

**DESCRIPTION** This routine allocates the resources needed to allow access to the MIB-II interface-group variables. This routine must be called before any interface variables can be accessed. The input parameter *pTrapRtn* is an optional pointer to a user-supplied SNMP trap generator. The input parameter *pTrapArg* is an optional argument to the trap generator. Only one trap generator is supported.

**RETURNS** OK, always.

**ERRNO** S\_m2Lib\_CANT\_CREATE\_IF\_SEM

**SEE ALSO** m2IfLib, m2IfGroupInfoGet(), m2IfTblEntryGet(), m2IfTblEntrySet(), m2IfDelete()

---

## **m2IfTblEntryGet()**

**NAME** m2IfTblEntryGet() – get a MIB-II interface-group table entry

**SYNOPSIS**

```
STATUS m2IfTblEntryGet
(
 int search, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_INTERFACETBL * pIfReqEntry /* pointer to requested interface entry */
)
```

***m2IfTblEntrySet()***

|                    |                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine maps the MIB-II interface index to the system's internal interface index. The <i>search</i> parameter is set to either <b>M2_EXACT_VALUE</b> or <b>M2_NEXT_VALUE</b> ; for a discussion of its use, see the manual entry for <b>m2Lib</b> . If the status of the interface has changed since it was last read, the user trap routine is called. |
| <b>RETURNS</b>     | OK, or ERROR if the input parameter is not specified, or a match is not found.                                                                                                                                                                                                                                                                               |
| <b>ERRNO</b>       | <b>S_m2Lib_INVALID_PARAMETER</b><br><b>S_m2Lib_ENTRY_NOT_FOUND</b>                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>m2IfLib</b> , <b>m2Lib</b> , <b>m2IfInit()</b> , <b>m2IfGroupInfoGet()</b> , <b>m2IfTblEntrySet()</b> , <b>m2IfDelete()</b>                                                                                                                                                                                                                               |

---

## ***m2IfTblEntrySet()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b><i>m2IfTblEntrySet()</i></b> – set the state of a MIB-II interface entry to UP or DOWN                                                                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre> STATUS m2IfTblEntrySet (     M2_INTERFACETBL * pIfTblEntry /* pointer to requested entry to change */ ) </pre>                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | This routine selects the interface specified in the input parameter <i>pIfTblEntry</i> and sets the interface to the requested state. It is the responsibility of the calling routine to set the interface index, and to make sure that the state specified in the <b>ifAdminStatus</b> field of the structure at <i>pIfTblEntry</i> is a valid MIB-II state, up(1) or down(2). |
| <b>RETURNS</b>     | OK, or ERROR if the input parameter is not specified, an interface is no longer valid, the interface index is incorrect, or the <i>ioctl()</i> command to the interface fails.                                                                                                                                                                                                  |
| <b>ERRNO</b>       | <b>S_m2Lib_INVALID_PARAMETER</b><br><b>S_m2Lib_ENTRY_NOT_FOUND</b><br><b>S_m2Lib_IF_CNFG_CHANGED</b>                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <b>m2IfLib</b> , <b>m2IfInit()</b> , <b>m2IfGroupInfoGet()</b> , <b>m2IfTblEntryGet()</b> , <b>m2IfDelete()</b>                                                                                                                                                                                                                                                                 |

---

## *m2Init()*

**NAME** *m2Init()* – initialize the SNMP MIB-2 library

**SYNOPSIS**

```
STATUS m2Init
(
 char * pMib2SysDescr, /* sysDescr */
 char * pMib2SysContact, /* sysContact */
 char * pMib2SysLocation, /* sysLocation */
 M2_OBJECTID * pMib2SysObjectId, /* sysObjectID */
 FUNCPTR pTrapRtn, /* link up/down -trap routine */
 void * pTrapArg, /* trap routine arg */
 int maxRouteTableSize /* max size of routing table */
)
```

**DESCRIPTION** This routine initializes the MIB-2 library by calling the initialization routines for each MIB-2 group. The parameters *pMib2SysDescr*, *pMib2SysContact*, *pMib2SysLocation*, and *pMib2SysObjectId* are passed directly to *m2SysInit()*; *pTrapRtn* and *pTrapArg* are passed directly to *m2IfInit()*; and *maxRouteTableSize* is passed to *m2IpInit()*.

**RETURNS** OK if successful, otherwise ERROR.

**SEE ALSO** *m2Lib*, *m2SysInit()*, *m2TcpInit()*, *m2UdpInit()*, *m2IcmpInit()*, *m2IfInit()*, *m2IpInit()*

---

## *m2IpAddrTblEntryGet()*

**NAME** *m2IpAddrTblEntryGet()* – get an IP MIB-II address entry

**SYNOPSIS**

```
STATUS m2IpAddrTblEntryGet
(
 int search, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_IPADRTBL * pIpAddrTblEntry /* ptr to requested IP address entry */
)
```

**DESCRIPTION** This routine traverses the IP address table and does an *M2\_EXACT\_VALUE* or a *M2\_NEXT\_VALUE* search based on the *search* parameter. The calling routine is responsible for supplying a valid MIB-II entry index in the input structure *pIpAddrTblEntry*. The index is the local IP address. The first entry in the table is retrieved by doing a NEXT search with the index field set to zero.

**RETURNS** OK, ERROR if the input parameter is not specified, or a match is not found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** m2IpLib, m2Lib, m2IpInit(), m2IpGroupInfoGet(), m2IpGroupInfoSet(),  
m2IpAtransTblEntrySet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(),  
m2IpDelete()

---

## **m2IpAtransTblEntryGet()**

**NAME** m2IpAtransTblEntryGet() – get a MIB-II ARP table entry

**SYNOPSIS**

```
STATUS m2IpAtransTblEntryGet
(
 int search, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_IPATRANSTBL * pReqIpAtEntry /* ptr to the requested ARP entry */
)
```

**DESCRIPTION** This routine traverses the ARP table and does an M2\_EXACT\_VALUE or a M2\_NEXT\_VALUE search based on the *search* parameter. The calling routine is responsible for supplying a valid MIB-II entry index in the input structure *pReqIpAtEntry*. The index is made up of the network interface index and the IP address corresponding to the physical address. The first entry in the table is retrieved by doing a NEXT search with the index fields set to zero.

**RETURNS** OK, ERROR if the input parameter is not specified, or a match is not found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** m2IpLib, m2Lib, m2IpInit(), m2IpGroupInfoGet(), m2IpGroupInfoSet(),  
m2IpAtransTblEntrySet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(),  
m2IpDelete()



---

## *m2IpAtransTblEntrySet()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>m2IpAtransTblEntrySet()</i> – add, modify, or delete a MIB-II ARP entry                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IpAtransTblEntrySet (     M2_IPATRANSTBL * pReqIpAtEntry /* pointer to MIB-II ARP entry */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | This routine traverses the ARP table for the entry specified in the parameter <i>pReqIpAtEntry</i> . An ARP entry can be added, modified, or deleted. A MIB-II entry index is specified by the destination IP address and the physical media address. A new ARP entry can be added by specifying all the fields in the parameter <i>pReqIpAtEntry</i> . An entry can be modified by specifying the MIB-II index and the field that is to be modified. An entry is deleted by specifying the index and setting the type field in the input parameter <i>pReqIpAtEntry</i> to the MIB-II value "invalid" (2). |
| <b>RETURNS</b>     | OK, or ERROR if the input parameter is not specified, the physical address is not specified for an add/modify request, or the <i>ioctl()</i> request to the ARP module fails.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>ERRNO</b>       | S_m2Lib_INVALID_PARAMETER<br>S_m2Lib_ARP_PHYSADDR_NOT_SPECIFIED                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <i>m2IpLib</i> , <i>m2IpInit()</i> , <i>m2IpGroupInfoGet()</i> , <i>m2IpGroupInfoSet()</i> ,<br><i>m2IpAddrTblEntryGet()</i> , <i>m2IpRouteTblEntryGet()</i> , <i>m2IpRouteTblEntrySet()</i> ,<br><i>m2IpDelete()</i>                                                                                                                                                                                                                                                                                                                                                                                       |

---

## *m2IpDelete()*

|                    |                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>m2IpDelete()</i> – delete all resources used to access the IP group                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IpDelete (void)</pre>                                                                                                                       |
| <b>DESCRIPTION</b> | This routine frees all the resources allocated when the IP group was initialized. The IP group should not be accessed after this routine has been called. |
| <b>RETURNS</b>     | OK, always.                                                                                                                                               |

SEE ALSO *m2IpLib*, *m2IpInit()*, *m2IpGroupInfoGet()*, *m2IpGroupInfoSet()*,  
*m2IpAddrTblEntryGet()*, *m2IpAtransTblEntrySet()*, *m2IpRouteTblEntryGet()*,  
*m2IpRouteTblEntrySet()*

---

## ***m2IpGroupInfoGet()***

NAME *m2IpGroupInfoGet()* – get the MIB-II IP-group scalar variables

SYNOPSIS 

```
STATUS m2IpGroupInfoGet
(
 M2_IP * pIpInfo /* pointer to IP MIB-II global group variables */
)
```

DESCRIPTION This routine fills in the IP structure at *pIpInfo* with the values of MIB-II IP global variables.

RETURNS OK, or ERROR if *pIpInfo* is not a valid pointer.

ERRNO **S\_m2Lib\_INVALID\_PARAMETER**

SEE ALSO *m2IpLib*, *m2IpInit()*, *m2IpGroupInfoSet()*, *m2IpAddrTblEntryGet()*,  
*m2IpAtransTblEntrySet()*, *m2IpRouteTblEntryGet()*, *m2IpRouteTblEntrySet()*,  
*m2IpDelete()*

---

## ***m2IpGroupInfoSet()***

NAME *m2IpGroupInfoSet()* – set MIB-II IP-group variables to new values

SYNOPSIS 

```
STATUS m2IpGroupInfoSet
(
 unsigned int varToSet, /* bit field used to set variables */
 M2_IP * pIpInfo /* ptr to the MIB-II IP group global variables */
)
```

DESCRIPTION This routine sets one or more variables in the IP group, as specified in the input structure *pIpInfo* and the bit field parameter *varToSet*.

RETURNS OK, or ERROR if *pIpInfo* is not a valid pointer, or *varToSet* has an invalid bit field.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
 S\_m2Lib\_INVALID\_VAR\_TO\_SET

**SEE ALSO** m2IpLib, m2IpInit(), m2IpGroupInfoGet(), m2IpAddrTblEntryGet(),  
 m2IpAtransTblEntrySet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(),  
 m2IpDelete()

## *m2IpInit()*

**NAME** m2IpInit() – initialize MIB-II IP-group access

**SYNOPSIS** STATUS m2IpInit  
 (  
     int maxRouteTableSize /\* max size of routing table \*/  
 )

**DESCRIPTION** This routine allocates the resources needed to allow access to the MIB-II IP variables. This routine must be called before any IP variables can be accessed. The parameter *maxRouteTableSize* is used to increase the default size of the MIB-II route table cache.

**RETURNS** OK, or ERROR if the route table or the route semaphore cannot be allocated.

**ERRNO** S\_m2Lib\_CANT\_CREATE\_ROUTE\_SEM

**SEE ALSO** m2IpLib, m2IpGroupInfoGet(), m2IpGroupInfoSet(), m2IpAddrTblEntryGet(),  
 m2IpAtransTblEntrySet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(),  
 m2IpDelete()

## *m2IpRouteTblEntryGet()*

**NAME** m2IpRouteTblEntryGet() – get a MIB-2 routing table entry

**SYNOPSIS** STATUS m2IpRouteTblEntryGet  
 (  
     int search, /\* M2\_EXACT\_VALUE or M2\_NEXT\_VALUE \*/  
     M2\_IPROUTETBL \* pIpRouteTblEntry /\* route table entry \*/  
 )

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | <p>This routine retrieves MIB-II information about an entry in the network routing table and returns it in the caller-supplied structure <i>pIpRouteTblEntry</i>.</p> <p>The routine compares routing table entries to the address specified by the <b>ipRouteDest</b> member of the <i>pIpRouteTblEntry</i> structure, and retrieves an entry chosen by the <i>search</i> type (<b>M2_EXACT_VALUE</b> or <b>M2_NEXT_VALUE</b>, as described in the manual entry for <b>m2Lib</b>).</p> |
| <b>RETURNS</b>     | OK if successful, otherwise ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>ERRNO</b>       | <b>S_m2Lib_INVALID_PARAMETER</b><br><b>S_m2Lib_ENTRY_NOT_FOUND</b>                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <b>m2IpLib</b> , <b>m2Lib</b> , <b>m2IpInit()</b> , <b>m2IpGroupInfoGet()</b> , <b>m2IpGroupInfoSet()</b> ,<br><b>m2IpAddrTblEntryGet()</b> , <b>m2IpRouteTblEntryGet()</b> , <b>m2IpRouteTblEntrySet()</b> ,<br><b>m2IpDelete()</b>                                                                                                                                                                                                                                                    |

---

## **m2IpRouteTblEntrySet()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>m2IpRouteTblEntrySet()</b> – set a MIB-II routing table entry                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IpRouteTblEntrySet (     int          varToSet,          /* variable to set */     M2_IPROUTETBL * pIpRouteTblEntry /* route table entry */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>DESCRIPTION</b> | <p>This routine adds, changes, or deletes a network routing table entry. The table entry to be modified is specified by the <b>ipRouteDest</b> and <b>ipRouteNextHop</b> members of the <i>pIpRouteTblEntry</i> structure.</p> <p>The <i>varToSet</i> parameter is a bit-field mask that specifies which values in the route table entry are to be set.</p> <p>If <i>varToSet</i> has the <b>M2_IP_ROUTE_TYPE</b> bit set and <b>ipRouteType</b> has the value of <b>M2_ROUTE_TYPE_INVALID</b>, then the the routing table entry is deleted.</p> <p>If <i>varToSet</i> has the either the <b>M2_IP_ROUTE_DEST</b> or <b>M2_IP_ROUTE_NEXT_HOP</b> bit set, then either a new route entry is added to the table or an existing route entry is changed.</p> |
| <b>RETURNS</b>     | OK if successful, otherwise ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>m2IpLib</b> , <b>m2IpInit()</b> , <b>m2IpGroupInfoGet()</b> , <b>m2IpGroupInfoSet()</b> ,<br><b>m2IpAddrTblEntryGet()</b> , <b>m2IpRouteTblEntryGet()</b> , <b>m2IpRouteTblEntrySet()</b> ,<br><b>m2IpDelete()</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

---

## *m2OspfAreaEntryGet()*

**NAME** *m2OspfAreaEntryGet()* – get an entry from the OSPF area table (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfAreaEntryGet
(
 int searchType, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_OSPF_AREA_ENTRY * pInfo /* ptr to area entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled with the contents of the area entry specified by *pInfo>ospfAreaId* and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** *ospfLib*

---

## *m2OspfAreaEntrySet()*

**NAME** *m2OspfAreaEntrySet()* – set values in an OSPF area entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfAreaEntrySet
(
 int varsToSet, /* flags specifying vars to set */
 M2_OSPF_AREA_ENTRY * pInfo /* ptr to area entry */
)
```

**DESCRIPTION** The area entry specified by *pInfo>ospfAreaId* will be updated with the values provided by *pInfo*. The *varsToSet* parameter indicates the fields to set and is a bitwise or of one or more of *M2\_OSPF\_AREA\_ID*, *M2\_OSPF\_AUTH\_TYPE*, and *M2\_OSPF\_IMPORT\_AS\_EXTERN*.

Note that the backbone area (0.0.0.0) is always present and does not need to be created explicitly. It is an error to use the *M2\_OSPF\_AREA\_ID* or *M2\_OSPF\_IMPORT\_AS\_EXTERN* flags with an area ID of 0.0.0.0.

**RETURNS** OK or ERROR.

**SEE ALSO** *ospfLib*

---

## *m2OspfAreaRangeEntryGet()*

**NAME** *m2OspfAreaRangeEntryGet()* – get an OSPF area range entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfAreaRangeEntryGet
(
 int searchType, /* M2_EXACT_VALUE */
 /* or M2_NEXT_VALUE */
 M2_OSPF_AREA_RANGE_ENTRY * pInfo /* ptr to area arange entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled in with the OSPF area range entry specified by *pInfo>ospfAreaRangeAreaID*, *pInfo>ospfAreaRangeNet*, and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** **ospfLib**

---

## *m2OspfAreaRangeEntrySet()*

**NAME** *m2OspfAreaRangeEntrySet()* – set values in an OSPF area range entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfAreaRangeEntrySet
(
 int varsToSet, /* flags specifying vars to set */
 M2_OSPF_AREA_RANGE_ENTRY * pInfo /* ptr to area range entry */
)
```

**DESCRIPTION** The OSPF area range entry specified by *pInfo>ospfAreaRangeAreaID* and *pInfo>ospfAreaRangeNet* is updated with the values provided in *pInfo*. The *varsToSet* parameter specifies the fields to set and is a bitwise or of one or more of M2\_OSPF\_AREA\_RANGE\_AREA\_ID, M2\_OSPF\_AREA\_RANGE\_NET, M2\_OSPF\_AREA\_RANGE\_MASK, and M2\_OSPF\_AREA\_RANGE\_STATUS.

**RETURNS** OK or ERROR.

**SEE ALSO** **ospfLib**

---

## *m2OspfGeneralGroupGet()*

- NAME** *m2OspfGeneralGroupGet()* – get values of OSPF general group objects (OSPF Opt.)
- SYNOPSIS**
- ```
STATUS m2OspfGeneralGroupGet
(
    M2_OSPF_GENERAL_GROUP * pInfo /* pointer to general group struct */
)
```
- DESCRIPTION** This routine fills in the structure pointed to by *pInfo* with the MIB-II values for the OSPF general group.
- RETURNS** OK, or ERROR if the get request fails.
- SEE ALSO** *ospfLib*

m2OspfGeneralGroupSet()

- NAME** *m2OspfGeneralGroupSet()* – set values of OSPF general group objects (OSPF Opt.)
- SYNOPSIS**
- ```
STATUS m2OspfGeneralGroupSet
(
 int varsToSet, /* flags specifying vars to set */
 M2_OSPF_GENERAL_GROUP * pInfo /* ptr to general group structure */
)
```
- DESCRIPTION** This routine sets the values of the OSPF general group objects. The variables to set are specified by a bitwise or of one or more of the flags *M2\_OSPF\_ROUTER\_ID*, *M2\_OSPF\_ADMIN\_STAT*, *M2\_OSPF\_AS\_BDR\_RTR\_STATUS*, and *M2\_OSPF\_TOS\_SUPPORT*, in the *varsToSet* parameter.
- RETURNS** OK or ERROR.
- SEE ALSO** *ospfLib*

---

## *m2OspfHostEntryGet()*

**NAME** *m2OspfHostEntryGet()* – get an OSPF host entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfHostEntryGet
(
 int searchType, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_OSPF_HOST_ENTRY * pInfo /* ptr to host entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled in with the entry specified by *pInfo>ospfHostIpAddress*, *pInfo>ospfHostTOS*, and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** *ospfLib*

---

## *m2OspfHostEntrySet()*

**NAME** *m2OspfHostEntrySet()* – set values in an OSPF host entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfHostEntrySet
(
 int varsToSet, /* flags specifying vars to set */
 M2_OSPF_HOST_ENTRY * pInfo /* ptr to host entry */
)
```

**DESCRIPTION** The OSPF host entry specified by *pInfo>ospfHostIpAddress* and *pInfo>ospfHostTOS* is updated with the values provided in *pInfo*. The *varsToSet* parameter indicates the fields to be set and is a bitwise or of one or more of *M2\_OSPF\_HOST\_IP\_ADDRESS*, *M2\_OSPF\_HOST\_TOS*, *M2\_OSPF\_HOST\_METRIC*, and *M2\_OSPF\_HOST\_STATUS*.

**RETURNS** OK or ERROR.

**SEE ALSO** *ospfLib*



---

## *m2OspfIfEntryGet()*

**NAME** *m2OspfIfEntryGet()* – get an OSPF interface entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfIfEntryGet
(
 int searchType, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_OSPF_IF_ENTRY * pInfo /* ptr ot interface entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled in with the entry specified by *pInfo>ospfIfIpAddress*, *pInfo>ospfAddressLessIf*, and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** *ospfLib*

---

## *m2OspfIfEntrySet()*

**NAME** *m2OspfIfEntrySet()* – set values in an OSPF interface entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfIfEntrySet
(
 int varsToSet, /* flags specifying vars to set */
 M2_OSPF_IF_ENTRY * pInfo /* ptr to interface entry */
)
```

**DESCRIPTION** This routine updates *pInfo>ospfAddressLessIf* with the contents of *pInfo*. The *varsToSet* parameter indicates the fields to set and is a bitwise or of one or more of:

```
M2_OSPF_IF_AREA_ID
M2_OSPF_IF_ADMIN_STAT
M2_OSPF_IF_RTR_PRIORITY
M2_OSPF_IF_TRANSIT_DELAY
M2_OSPF_IF_RETRANS_INTERVAL
M2_OSPF_IF_HELLO_INTERVAL
M2_OSPF_IF_RTR_DEAD_INTERVAL
M2_OSPF_IF_POLL_INTERVAL
M2_OSPF_IF_AUTH_KEY
```

**RETURNS** OK or ERROR.

SEE ALSO **ospfLib**

---

## ***m2OspfIfMetricEntryGet()***

**NAME** *m2OspfIfMetricEntryGet()* – get an OSPF interface metric entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfIfMetricEntryGet
(
 int searchType, /* M2_EXACT_VALUE */
 /* or M2_NEXT_VALUE */
 M2_OSPF_IF_METRIC_ENTRY * pInfo /* ptr to interface metric entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled in with the entry specified by **pInfo>ospfIfMetricIpAddress**, **pInfo>ospfIfMetricAddressLessIf**, **pInfo>ospfIfMetricTOS**, and *searchType*.

**RETURNS** OK or ERROR.

SEE ALSO **ospfLib**

---

## ***m2OspfIfMetricEntrySet()***

**NAME** *m2OspfIfMetricEntrySet()* – set OSPF interface metric entry values (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfIfMetricEntrySet
(
 int varsToSet, /* flags specifying vars to set */
 M2_OSPF_IF_METRIC_ENTRY * pInfo /* ptr to interface metric entry */
)
```

**DESCRIPTION** The fields of the OSPF interface metric entry specified by **pInfo>ospfIfMetricIpAddress**, **pInfo>ospfIfMetricAddress**, and **pInfo>ospfIfMetricTOS** is updated with the contents of *pInfo*. The *varsToSet* parameter indicates the fields to set and is a bitwise or of one or more of **M2\_OSPF\_IF\_METRIC\_METRIC** or **M2\_OSPF\_IF\_METRIC\_STATUS**.

**RETURNS** OK or ERROR.

SEE ALSO **ospfLib**

---

## *m2OspfLsdbEntryGet()*

**NAME** *m2OspfLsdbEntryGet()* – get an OSPF link state database entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfLsdbEntryGet
(
 int searchType, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_OSPF_LSDB_ENTRY * pInfo /* link state database entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled in with the entry specified by *pInfo>ospfLsdbAreaId*, *pInfo>ospfLsdbType*, and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** *ospfLib*

---

## *m2OspfNbrEntryGet()*

**NAME** *m2OspfNbrEntryGet()* – get an OSPF neighbor entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfNbrEntryGet
(
 int searchType, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_OSPF_NBR_ENTRY * pInfo /* ptr to neighbor entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled in with the contents of the OSPF neighbor entry specified by *pInfo>ospfNbrIpAddr*, *pInfo>ospfNbrAddressLessIndex* and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** *ospfLib*

---

## ***m2OspfNbrEntrySet()***

**NAME** *m2OspfNbrEntrySet()* – set values in an OSPF neighbor entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfNbrEntrySet
(
 int varsToSet, /* flags specifying vars to set */
 M2_OSPF_NBR_ENTRY * pInfo /* ptr to neighbor entry */
)
```

**DESCRIPTION** The OSPF neighbor entry specified by **pInfo>ospfNbrIpAddr** and **pInfo>ospfNbrAddressLessIndex** is updated with the contents of *pInfo*. The *varsToSet* parameter indicates the fields to set, which can be **M2\_OSPF\_NBMA\_NBR\_STATUS**.

**RETURNS** OK or ERROR.

**SEE ALSO** **ospfLib**

---

## ***m2OspfStubAreaEntryGet()***

**NAME** *m2OspfStubAreaEntryGet()* – get an OSPF stub area entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfStubAreaEntryGet
(
 int searchType, /* M2_EXACT_VALUE */
 /* or M2_NEXT_VALUE */
 M2_OSPF_STUB_AREA_ENTRY * pInfo /* ptr to stub area entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled with the contents of the stub area entry specified by **pInfo>ospfStubAreaID**, **pInfo>ospfStubTOS** and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** **ospfLib**

---

## *m2OspfStubAreaEntrySet()*

**NAME** *m2OspfStubAreaEntrySet()* – set values in an OSPF stub area entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfStubAreaEntrySet
(
 int varsToSet, /* flags specifying vars to set */
 M2_OSPF_STUB_AREA_ENTRY * pInfo /* ptr to stub area entry */
)
```

**DESCRIPTION** The stub area entry specified by **pInfo>ospfStubAreaID** and **pInfo>ospfStubTOS** is updated with the values provided in *pInfo*. The *varsToSet* parameter indicates the fields to be modified and is a bitwise or of one or more of **M2\_OSPF\_STUB\_AREA\_ID**, **M2\_OSPF\_STUB\_TOS**, **M2\_OSPF\_STUB\_METRIC**, and **M2\_OSPF\_STUB\_STATUS**.

**RETURNS** OK or ERROR.

**SEE ALSO** **ospfLib**

---

## *m2OspfVirtIfEntryGet()*

**NAME** *m2OspfVirtIfEntryGet()* – get an OSPF virtual interface entry (OSPF Opt.)

**SYNOPSIS**

```
STATUS m2OspfVirtIfEntryGet
(
 int searchType, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_OSPF_VIRT_IF_ENTRY * pInfo /* ptr to virtual interface entry */
)
```

**DESCRIPTION** The structure pointed to by *pInfo* is filled in with the contents of the OSPF virtual interface entry specified by **pInfo>ospfVirtIfAreaID**, **pInfo>ospfVirtIfNeighbor** and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** **ospfLib**

---

## ***m2OspfVirtIfEntrySet()***

- NAME** *m2OspfVirtIfEntrySet()* – set OSPF virtual interface entry values (OSPF Opt.)
- SYNOPSIS**
- ```
STATUS m2OspfVirtIfEntrySet
(
    int                varsToSet, /* flags specifying vars to set */
    M2_OSPF_VIRT_IF_ENTRY * pInfo    /* ptr to virtual interface entry */
)
```
- DESCRIPTION**
- The OSPF virtual interface entry specified by **pInfo>ospfVirtIfAreaID** and **pInfo>ospfVirtIfNeighbor** is updated with the contents of *pInfo*. The *varsToSet* parameter indicates the fields to be modified and is a bitwise or of one or more of:
- M2_OSPF_VIRT_IF_AREA_ID
 - M2_OSPF_VIRT_IF_NEIGHBOR
 - M2_OSPF_VIRT_IF_TRANSIT_DELAY
 - M2_OSPF_VIRT_IF_HELLO_INTERVAL
 - M2_OSPF_VIRT_IF_RTR_DEAD_INTERVAL
 - M2_OSPF_VIRT_IF_STATUS
 - M2_OSPF_VIRT_IF_AUTH_KEY
- RETURNS** OK or ERROR.
- SEE ALSO** *ospfLib*

m2OspfVirtNbrEntryGet()

- NAME** *m2OspfVirtNbrEntryGet()* – get an OSPF virtual neighbor entry (OSPF Opt.)
- SYNOPSIS**
- ```
STATUS m2OspfVirtNbrEntryGet
(
 int searchType, /* M2_EXACT_VALUE */
 /* or M2_NEXT_VALUE */
 M2_OSPF_VIRT_NBR_ENTRY * pInfo /* ptr to virtual neighbor entry */
)
```
- DESCRIPTION**
- The structure pointed to by *pInfo* is filled in with the contents of the OSPF virtual neighbor entry specified by **pInfo>ospfVirtNbrArea**, **pInfo>ospfVirtNbrRtrId**, and *searchType*.

**RETURNS** OK or ERROR.

**SEE ALSO** `ospfLib`

---

## *m2SysDelete()*

**NAME** *m2SysDelete()* – delete resources used to access the MIB-II system group

**SYNOPSIS** `STATUS m2SysDelete (void)`

**DESCRIPTION** This routine frees all the resources allocated at the time the group was initialized. Do not access the system group after calling this routine.

**RETURNS** OK, always.

**SEE ALSO** `m2SysLib`, `m2SysInit()`, `m2SysGroupInfoGet()`, `m2SysGroupInfoSet()`.

---

## *m2SysGroupInfoGet()*

**NAME** *m2SysGroupInfoGet()* – get system-group MIB-II variables

**SYNOPSIS** `STATUS m2SysGroupInfoGet`  
(  
    *M2\_SYSTEM* \* *pSysInfo* /\* pointer to MIB-II system group structure \*/  
)

**DESCRIPTION** This routine fills in the structure at *pSysInfo* with the values of MIB-II system-group variables.

**RETURNS** OK, or ERROR if *pSysInfo* is not a valid pointer.

**ERRNO** `S_m2Lib_INVALID_PARAMETER`

**SEE ALSO** `m2SysLib`, `m2SysInit()`, `m2SysGroupInfoSet()`, `m2SysDelete()`

---

## ***m2SysGroupInfoSet()***

|                    |                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>m2SysGroupInfoSet()</i> – set system-group MIB-II variables to new values                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS m2SysGroupInfoSet (     unsigned int varToSet, /* bit field of variables to set */     M2_SYSTEM * pSysInfo /* pointer to the system structure */ )</pre> |
| <b>DESCRIPTION</b> | This routine sets one or more variables in the system group as specified in the input structure at <i>pSysInfo</i> and the bit field parameter <i>varToSet</i> .      |
| <b>RETURNS</b>     | OK, or ERROR if <i>pSysInfo</i> is not a valid pointer, or <i>varToSet</i> has an invalid bit field.                                                                  |
| <b>ERRNO</b>       | S_m2Lib_INVALID_PARAMETER<br>S_m2Lib_INVALID_VAR_TO_SET                                                                                                               |
| <b>SEE ALSO</b>    | <i>m2SysLib</i> , <i>m2SysInit()</i> , <i>m2SysGroupInfoGet()</i> , <i>m2SysDelete()</i>                                                                              |

---

## ***m2SysInit()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>m2SysInit()</i> – initialize MIB-II system-group routines                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS m2SysInit (     char *      pMib2SysDescr, /* pointer to MIB-2 sysDescr */     char *      pMib2SysContact, /* pointer to MIB-2 sysContact */     char *      pMib2SysLocation, /* pointer to MIB-2 sysLocation */     M2_OBJECTID * pObjectId /* pointer to MIB-2 ObjectId */ )</pre>                                                                                                                                                    |
| <b>DESCRIPTION</b> | This routine allocates the resources needed to allow access to the system-group MIB-II variables. This routine must be called before any system-group variables can be accessed. The input parameters <i>pMib2SysDescr</i> , <i>pMib2SysContact</i> , <i>pMib2SysLocation</i> , and <i>pObjectId</i> are optional. The parameters <i>pMib2SysDescr</i> , <i>pObjectId</i> are read only, as specified by MIB-II, and can be set only by this routine. |
| <b>RETURNS</b>     | OK, always.                                                                                                                                                                                                                                                                                                                                                                                                                                           |



**ERRNO** S\_m2Lib\_CANT\_CREATE\_SYS\_SEM

**SEE ALSO** m2SysLib, m2SysGroupInfoGet(), m2SysGroupInfoSet(), m2SysDelete()

---

## *m2TcpConnEntryGet()*

**NAME** *m2TcpConnEntryGet()* – get a MIB-II TCP connection table entry

**SYNOPSIS**

```
STATUS m2TcpConnEntryGet
(
 int search, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
 M2_TCPCONNTBL * pReqTcpConnEntry /* input = Index, Output = Entry */
)
```

**DESCRIPTION** This routine traverses the TCP table of users and does an M2\_EXACT\_VALUE or a M2\_NEXT\_VALUE search based on the *search* parameter (see **m2Lib**). The calling routine is responsible for supplying a valid MIB-II entry index in the input structure *pReqTcpConnEntry*. The index is made up of the local IP address, the local port number, the remote IP address, and the remote port. The first entry in the table is retrieved by doing a M2\_NEXT\_VALUE search with the index fields set to zero.

**RETURNS** OK, or ERROR if the input parameter is not specified or a match is not found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** m2TcpLib, m2Lib, m2TcpInit(), m2TcpGroupInfoGet(), m2TcpConnEntrySet(), m2TcpDelete()

---

## *m2TcpConnEntrySet()*

**NAME** *m2TcpConnEntrySet()* – set a TCP connection to the closed state

**SYNOPSIS**

```
STATUS m2TcpConnEntrySet
(
 M2_TCPCONNTBL * pReqTcpConnEntry /* pointer to TCP connection to close */
)
```

***m2TcpDelete()***

- DESCRIPTION** This routine traverses the TCP connection table and searches for the connection specified by the input parameter *pReqTcpConnEntry*. The calling routine is responsible for providing a valid index as the input parameter *pReqTcpConnEntry*. The index is made up of the local IP address, the local port number, the remote IP address, and the remote port. This call can only succeed if the connection is in the MIB-II state "deleteTCB" (12). If a match is found, the socket associated with the TCP connection is closed.
- RETURNS** OK, or ERROR if the input parameter is invalid, the state of the connection specified at *pReqTcpConnEntry* is not "closed," the specified connection is not found, a socket is not associated with the connection, or the *close()* call fails.
- SEE ALSO** *m2TcpLib*, *m2TcpInit()*, *m2TcpGroupInfoGet()*, *m2TcpConnEntryGet()*, *m2TcpDelete()*

***m2TcpDelete()***

- NAME** *m2TcpDelete()* – delete all resources used to access the TCP group
- SYNOPSIS** `STATUS m2TcpDelete (void)`
- DESCRIPTION** This routine frees all the resources allocated at the time the group was initialized. The TCP group should not be accessed after this routine has been called.
- RETURNS** OK, always.
- SEE ALSO** *m2TcpLib*, *m2TcpInit()*, *m2TcpGroupInfoGet()*, *m2TcpConnEntryGet()*, *m2TcpConnEntrySet()*

***m2TcpGroupInfoGet()***

- NAME** *m2TcpGroupInfoGet()* – get MIB-II TCP-group scalar variables
- SYNOPSIS** `STATUS m2TcpGroupInfoGet  
(  
    M2_TCPIINFO * pTcpInfo /* pointer to the TCP group structure */  
)`
- DESCRIPTION** This routine fills in the TCP structure pointed to by *pTcpInfo* with the values of MIB-II TCP-group scalar variables.

**RETURNS** OK, or ERROR if *pTcpInfo* is not a valid pointer.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER

**SEE ALSO** m2TcpLib, m2TcpInit(), m2TcpConnEntryGet(), m2TcpConnEntrySet(), m2TcpDelete()

---

## **m2TcpInit()**

**NAME** *m2TcpInit()* – initialize MIB-II TCP-group access

**SYNOPSIS** STATUS m2TcpInit (void)

**DESCRIPTION** This routine allocates the resources needed to allow access to the TCP MIB-II variables. This routine must be called before any TCP variables can be accessed.

**RETURNS** OK, always.

**SEE ALSO** m2TcpLib, m2TcpGroupInfoGet(), m2TcpConnEntryGet(), m2TcpConnEntrySet(), m2TcpDelete()

---

## **m2UdpDelete()**

**NAME** *m2UdpDelete()* – delete all resources used to access the UDP group

**SYNOPSIS** STATUS m2UdpDelete (void)

**DESCRIPTION** This routine frees all the resources allocated at the time the group was initialized. The UDP group should not be accessed after this routine has been called.

**RETURNS** OK, always.

**SEE ALSO** m2UdpLib, m2UdpInit(), m2UdpGroupInfoGet(), m2UdpTblEntryGet()

---

## ***m2UdpGroupInfoGet()***

**NAME** *m2UdpGroupInfoGet()* – get MIB-II UDP-group scalar variables

**SYNOPSIS**

```
STATUS m2UdpGroupInfoGet
(
 M2_UDP * pUdpInfo /* pointer to the UDP group structure */
)
```

**DESCRIPTION** This routine fills in the UDP structure at *pUdpInfo* with the MIB-II UDP scalar variables.

**RETURNS** OK, or ERROR if *pUdpInfo* is not a valid pointer.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER

**SEE ALSO** *m2UdpLib*, *m2UdpInit()*, *m2UdpTblEntryGet()*, *m2UdpDelete()*

---

## ***m2UdpInit()***

**NAME** *m2UdpInit()* – initialize MIB-II UDP-group access

**SYNOPSIS**

```
STATUS m2UdpInit (void)
```

**DESCRIPTION** This routine allocates the resources needed to allow access to the UDP MIB-II variables. This routine must be called before any UDP variables can be accessed.

**RETURNS** OK, always.

**SEE ALSO** *m2UdpLib*, *m2UdpGroupInfoGet()*, *m2UdpTblEntryGet()*, *m2UdpDelete()*

---

## ***m2UdpTblEntryGet()***

**NAME** *m2UdpTblEntryGet()* – get a UDP MIB-II entry from the UDP list of listeners

**SYNOPSIS**

```
STATUS m2UdpTblEntryGet
(
 int search, /* M2_EXACT_VALUE or M2_NEXT_VALUE */
```

```
M2_UDPTBL * pUdpEntry /* ptr to the requested entry with index */
)
```

**DESCRIPTION** This routine traverses the UDP table of listeners and does an **M2\_EXACT\_VALUE** or a **M2\_NEXT\_VALUE** search based on the *search* parameter. The calling routine is responsible for supplying a valid MIB-II entry index in the input structure *pUdpEntry*. The index is made up of the IP address and the local port number. The first entry in the table is retrieved by doing a **M2\_NEXT\_VALUE** search with the index fields set to zero.

**RETURNS** OK, or ERROR if the input parameter is not specified or a match is not found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** m2UdpLib, m2Lib, m2UdpInit(), m2UdpGroupInfoGet(), m2UdpDelete()

## ***m68302SioInit()***

**NAME** *m68302SioInit()* – initialize a **M68302\_CP**

**SYNOPSIS**

```
void m68302SioInit
(
 M68302_CP * pCp
)
```

**DESCRIPTION** This routine initializes the driver function pointers and then resets the chip to a quiescent state. The BSP must already have initialized all the device addresses and the **baudFreq** fields in the **M68302\_CP** structure before passing it to this routine. The routine resets the device and initializes everything to support polled mode (if possible), but does not enable interrupts.

**RETURNS** N/A

**SEE ALSO** **m68302Sio**

## ***m68302SioInit2()***

**NAME** *m68302SioInit2()* – initialize a **M68302\_CP** (part 2)

**SYNOPSIS**

```
void m68302SioInit2
(
 M68302_CP * pCp
)
```

**DESCRIPTION** Enables interrupt mode of operation.

**RETURNS** N/A

**SEE ALSO** **m68302Sio**

---

## ***m68332DevInit()***

**NAME** *m68332DevInit()* – initialize the SCC

**SYNOPSIS**

```
void m68332DevInit
(
 M68332_CHAN * pChan
)
```

**DESCRIPTION** This initializes the chip to a quiescent state.

**RETURNS** N/A

**SEE ALSO** **m68332Sio**

---

## *m68332Int()*

**NAME** *m68332Int()* – handle an SCC interrupt

**SYNOPSIS**

```
void m68332Int
(
 M68332_CHAN * pChan
)
```

**DESCRIPTION** This routine handles SCC interrupts.

**RETURNS** N/A

**SEE ALSO** *m68332Sio*

---

## *m68360DevInit()*

**NAME** *m68360DevInit()* – initialize the SCC

**SYNOPSIS**

```
void m68360DevInit
(
 M68360_CHAN * pChan
)
```

**DESCRIPTION** This routine is called to initialize the chip to a quiescent state.

**SEE ALSO** *m68360Sio*

---

## *m68360Int()*

**NAME** *m68360Int()* – handle an SCC interrupt

**SYNOPSIS**

```
void m68360Int
(
 M68360_CHAN * pChan
)
```

**DESCRIPTION** This routine gets called to handle SCC interrupts.

**SEE ALSO** **m68360Sio**

---

## ***m68562HrdInit()***

**NAME** ***m68562HrdInit()*** – initialize the DUSCC

**SYNOPSIS**

```
void m68562HrdInit
(
 M68562_QUSART * pQusart
)
```

**DESCRIPTION** The BSP must have already initialized all the device addresses, etc in **M68562\_DUSART** structure. This routine resets the chip in a quiescent state.

**SEE ALSO** **m68562Sio**

---

## ***m68562RxInt()***

**NAME** ***m68562RxInt()*** – handle a receiver interrupt

**SYNOPSIS**

```
void m68562RxInt
(
 M68562_CHAN * pChan
)
```

**RETURNS** N/A

**SEE ALSO** **m68562Sio**



---

## ***m68562RxTxErrInt()***

**NAME** *m68562RxTxErrInt()* – handle a receiver/transmitter error interrupt

**SYNOPSIS**

```
void m68562RxTxErrInt
(
 M68562_CHAN * pChan
)
```

**DESCRIPTION** Only the receive overrun condition is handled.

**RETURNS** N/A

**SEE ALSO** **m68562Sio**

---

## ***m68562TxInt()***

**NAME** *m68562TxInt()* – handle a transmitter interrupt

**SYNOPSIS**

```
void m68562TxInt
(
 M68562_CHAN * pChan
)
```

**DESCRIPTION** If there is another character to be transmitted, it sends it. If not, or if a device has never been created for this channel, disable the interrupt.

**RETURNS** N/A

**SEE ALSO** **m68562Sio**

---

## ***m68681Acr()***

- NAME** *m68681Acr()* – return the contents of the DUART auxiliary control register
- SYNOPSIS**
- ```
UCHAR m68681Acr
(
    M68681_DUART * pDuart
)
```
- DESCRIPTION** This routine returns the contents of the auxilliary control register (ACR). The ACR is not directly readable; a copy of the last value written is kept in the DUART data structure.
- RETURNS** The contents of the auxilliary control register.
- SEE ALSO** **m68681Sio**

m68681AcrSetClr()

- NAME** *m68681AcrSetClr()* – set and clear bits in the DUART auxiliary control register
- SYNOPSIS**
- ```
void m68681AcrSetClr
(
 M68681_DUART * pDuart,
 UCHAR setBits, /* which bits to set in the ACR */
 UCHAR clearBits /* which bits to clear in the ACR */
)
```
- DESCRIPTION** This routine sets and clears bits in the DUART auxiliary control register (ACR). It sets and clears bits in a local copy of the ACR, then writes that local copy to the DUART. This means that all changes to the ACR must be performed by this routine. Any direct changes to the ACR are lost the next time this routine is called.
- Set has priority over clear. Thus you can use this routine to update multiple bit fields by specifying the field mask as the clear bits.
- RETURNS** N/A
- SEE ALSO** **m68681Sio**

---

## *m68681DevInit()*

|                    |                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>m68681DevInit()</i> – initialize a M68681_DUART                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>void m68681DevInit (     M68681_DUART * pDuart )</pre>                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | The BSP must already have initialized all the device addresses and register pointers in the M68681_DUART structure as described in <b>m68681Sio</b> . This routine initializes some transmitter and receiver status values to be used in the interrupt mask register and then resets the chip to a quiescent state. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>m68681Sio</b>                                                                                                                                                                                                                                                                                                    |

---

## *m68681DevInit2()*

|                    |                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>m68681DevInit2()</i> – initialize a M68681_DUART, part 2                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>void m68681DevInit2 (     M68681_DUART * pDuart )</pre>                                                                                                                  |
| <b>DESCRIPTION</b> | This routine is called as part of <i>sysSerialHwInit2()</i> . It tells the driver that interrupt vectors are connected and that it is safe to allow interrupts to be enabled. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>m68681Sio</b>                                                                                                                                                              |

---

## ***m68681Imr()***

- NAME** *m68681Imr()* – return the current contents of the DUART interrupt-mask register
- SYNOPSIS**
- ```
UCHAR m68681Imr
(
    M68681_DUART * pDuart
)
```
- DESCRIPTION** This routine returns the contents of the interrupt-mask register (IMR). The IMR is not directly readable; a copy of the last value written is kept in the DUART data structure.
- RETURNS** The contents of the interrupt-mask register.
- SEE ALSO** **m68681Sio**

m68681ImrSetClr()

- NAME** *m68681ImrSetClr()* – set and clear bits in the DUART interrupt-mask register
- SYNOPSIS**
- ```
void m68681ImrSetClr
(
 M68681_DUART * pDuart,
 UCHAR setBits, /* which bits to set in the IMR */
 UCHAR clearBits /* which bits to clear in the IMR */
)
```
- DESCRIPTION** This routine sets and clears bits in the DUART interrupt-mask register (IMR). It sets and clears bits in a local copy of the IMR, then writes that local copy to the DUART. This means that all changes to the IMR must be performed by this routine. Any direct changes to the IMR are lost the next time this routine is called.
- Set has priority over clear. Thus you can use this routine to update multiple bit fields by specifying the field mask as the clear bits.
- RETURNS** N/A
- SEE ALSO** **m68681Sio**

---

## *m68681Int()*

|                    |                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>m68681Int()</i> – handle all DUART interrupts in one vector                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>void m68681Int (     M68681_DUART * pDuart )</pre>                                                                                                                    |
| <b>DESCRIPTION</b> | This routine handles all interrupts in a single interrupt vector. It identifies and services each interrupting source in turn, using edge-sensitive interrupt controllers. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                        |
| <b>SEE ALSO</b>    | <i>m68681Sio</i>                                                                                                                                                           |

---

## *m68681Opcr()*

|                    |                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>m68681Opcr()</i> – return the state of the DUART output port configuration register                                                                                               |
| <b>SYNOPSIS</b>    | <pre>UCHAR m68681Opcr (     M68681_DUART * pDuart )</pre>                                                                                                                            |
| <b>DESCRIPTION</b> | This routine returns the state of the output port configuration register (OPCR) from the saved copy in the DUART data structure. The actual OPCR contents are not directly readable. |
| <b>RETURNS</b>     | The state of the output port configuration register.                                                                                                                                 |
| <b>SEE ALSO</b>    | <i>m68681Sio</i>                                                                                                                                                                     |

## *m68681OpcrSetClr()*

**NAME** *m68681OpcrSetClr()* – set and clear bits in the DUART output port configuration register

**SYNOPSIS**

```
void m68681OpcrSetClr
(
 M68681_DUART * pDuart,
 UCHAR setBits, /* which bits to set in the OPCR */
 UCHAR clearBits /* which bits to clear in the OPCR */
)
```

**DESCRIPTION** This routine sets and clears bits in the DUART output port configuration register (OPCR). It sets and clears bits in a local copy of the OPCR, then writes that local copy to the DUART. This means that all changes to the OPCR must be performed by this routine. Any direct changes to the OPCR are lost the next time this routine is called.

Set has priority over clear. Thus you can use this routine to update multiple bit fields by specifying the field mask as the clear bits.

**RETURNS** N/A

**SEE ALSO** *m68681Sio*

---

## *m68681Opr()*

**NAME** *m68681Opr()* – return the current state of the DUART output port register

**SYNOPSIS**

```
UCHAR m68681Opr
(
 M68681_DUART * pDuart
)
```

**DESCRIPTION** This routine returns the current state of the output port register (OPR) from the saved copy in the DUART data structure. The actual OPR contents are not directly readable.

**RETURNS** The current state of the output port register.

**SEE ALSO** *m68681Sio*

---

## *m68681OprSetClr()*

**NAME** *m68681OprSetClr()* – set and clear bits in the DUART output port register

**SYNOPSIS**

```
void m68681OprSetClr
(
 M68681_DUART * pDuart,
 UCHAR setBits, /* which bits to set in the OPR */
 UCHAR clearBits /* which bits to clear in the OPR */
)
```

**DESCRIPTION** This routine sets and clears bits in the DUART output port register (OPR). It sets and clears bits in a local copy of the OPR, then writes that local copy to the DUART. This means that all changes to the OPR must be performed by this routine. Any direct changes to the OPR are lost the next time this routine is called.

Set has priority over clear. Thus you can use this routine to update multiple bit fields by specifying the field mask as the clear bits.

**RETURNS** N/A

**SEE ALSO** *m68681Sio*

---

## *m68901DevInit()*

**NAME** *m68901DevInit()* – initialize a M68901\_CHAN structure

**SYNOPSIS**

```
void m68901DevInit
(
 M68901_CHAN * pChan
)
```

**DESCRIPTION** This routine initializes the driver function pointers and then resets the chip to a quiescent state. The BSP must have already initialized all the device addresses and the **baudFreq** fields in the M68901\_CHAN structure before passing it to this routine.

**RETURNS** N/A

**SEE ALSO** *m68901Sio*

## *malloc()*

|                    |                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>malloc()</i> – allocate a block of memory from the system memory partition (ANSI)                                                                             |
| <b>SYNOPSIS</b>    | <pre>void *malloc (     size_t nBytes /* number of bytes to allocate */ )</pre>                                                                                  |
| <b>DESCRIPTION</b> | This routine allocates a block of memory from the free list. The size of the block will be equal to or greater than <i>nBytes</i> .                              |
| <b>RETURNS</b>     | A pointer to the allocated block of memory, or a null pointer if there is an error.                                                                              |
| <b>SEE ALSO</b>    | <b>memPartLib</b> , <i>American National Standard for Information Systems – Programming Language – C</i> , ANSI X3.159-1989: <i>General Utilities (stdlib.h)</i> |

---

## *masterIoInit()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>masterIoInit()</i> – create the IPC mechanism at the SNMP master agent                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS masterIoInit ( void )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine, called from <i>snmpIoInit()</i>, creates the SNMP master agent side of the inter-process communication (IPC) mechanism used to carry messages between subagents and the master agent. In this implementation, <i>masterIoInit()</i> creates a single message queue. The identity of this message queue is hard coded into every subagent. The subagent puts a message on this queue when it needs to send a message to the master agent.</p> <p>The message queue created by <i>masterIoInit()</i> is monitored by <b>tMonQue</b>. The <b>tMonQue</b> task is one of the two tasks used to implement the SNMP master agent. The purpose of <b>tMonQue</b> is to note which messages in its queue are registration requests and which are responses to queries. If the message is a subagent registration request, <b>tMonQue</b> handles the request and sends a message back to the subagent telling it whether the registration was successful or not.</p> <p>If the message is a response to a query, <b>tMonQue</b> transfers the message to the message queue monitored by <b>tSnmpd</b>. The <b>tSnmpd</b> task then encodes the response in an SNMP packet and transmits the packet over a socket to the SNMP manager.</p> |



Although the shipped version of this function uses message queues as the IPC between the master agent and its subagents, the IPC mechanism is isolated to the relatively small number of functions defined in **masterIoLib**. Thus, if necessary, you should have little trouble porting the code to use an IPC more suitable to your transport needs.

For example, you could use sockets instead of message queues. However, if you decide to change the IPC mechanism, you must do so both in the master agent and in its subagents. This means that you must also modify the functions defined in **saIoLib**, the library that defines the agent side of the IPC mechanism.

**RETURNS** OK or ERROR.

**SEE ALSO** **masterIoLib**

---

## **masterIoWrite()**

**NAME** *masterIoWrite()* – send the encoded buffer to the subagent

**SYNOPSIS**

```
STATUS masterIoWrite
(
 EBUFFER_T * pBuf, /* reply message to be sent */
 PTR_T saId, /* subagent address */
 INT_32_T flg /* type of message */
)
```

**DESCRIPTION** This routine transmits the byte array at *pBuf* to the subagent at *saId*. This routine is called from a wide variety of functions in the master agent. For example, *masterIpcSend()* calls this routine when it needs to query the subagent about one of the MIB variables it manages. Likewise, the *masterIpcAyt()* function calls this routine when needs to check the IPC link status. Similarly, *snmpQueMonitor()* calls this routine to tell the agent the results of a registration or deregistration request.

The master agent uses the value *flg* to specify the general nature of the message it is writing to the subagent, which partially determines how the subagent responds. For example, when the master agent is responding to the subagent after successfully handling its registration request, the master agent uses a *flg* value of **REG\_COMPLETE**. When the master agent does an "are you there" check, it specifies a *flg* value of **IPC\_AYT**. **REG\_COMPLETE** and **IPC\_AYT** are the only currently valid *flg* values.

**RETURNS** OK or ERROR.

**SEE ALSO** **masterIoLib**

## *masterIpcAyt()*

- NAME** *masterIpcAyt()* – check the status of the IPC link
- SYNOPSIS**
- ```
INT_32_T masterIpcAyt
(
    PTR_T ipchandle /* pointer to IPC handle */
)
```
- DESCRIPTION** This is an "are you there" routine. The SNMP master agent calls this routine whenever it needs to do a status check on the IPC link to the address *ipchandle*. This routine puts a null-buffer message of type `IPC_AYT` on the subagent's message queue. If the subagent replies with a message of the same type, the link is considered active.
- RETURNS** 0, if the link is inactive; 1, if the link is inactive
- SEE ALSO** *masterIoLib*
-

masterIpcComp()

- NAME** *masterIpcComp()* – transmit a completion of transmission message
- SYNOPSIS**
- ```
void masterIpcComp
(
 OCTET_T opcode, /* this specifies what needs to be done */
 EBUFFER_T * ebuf, /* reply message to be sent */
 VBL_T * vblist, /* list of varbinds that the message contained */
 PTR_T ipchandle /* subagent address */
)
```
- DESCRIPTION** If the SNMP master agent uses *snmpMasterHandlerAsync()* to process a subagent's unsolicited control message (such as a registration request), it uses *masterIpcComp()* to complete processing for the message. In the current implementation, this means telling the subagent the completion status of a registration or deregistration request. However, you can rewrite this function to implement a broader range of responses (such as forwarding traps to the SNMP manager).
- When the master agent calls this routine, it uses *opcode* to indicate the processing status of the message. If the status indicates an error, *masterIpcComp()* drops the packet. If the status indicates success, the master agent uses the *ebuf* parameter to pass in a message for the subagent at *ipchandle*. Internally, *masterIpcComp()* calls *masterIoWrite()* to forward

the message to the specified subagent. If this message is the response to a successful registration request, it contains the group ID for the MIB variables just added to the master agent's MIB tree. The subagent needs this group ID for any deregistration request it might send later. It also uses this ID to register instances of the object just registered.

If the *opcode* is a value of 1 or greater (up to and including 127), the master agent uses the *vblist* parameter to pass in a varbind list that it extracted from the control message. In the current implementation, the *masterIpcComp()* routine does nothing with the message and returns. However, you could modify *masterIpcComp()* to process the message according to the value specified by *opcode*. For example, if *opcode* indicates a trap, you could forward the information at *vblist* to the SNMP manager.

Currently, **subagent.h** defines symbolic constants for opcodes 1 through 12 (with opcode 11, **SA\_TRAP\_REQUEST**, reserved for trap requests). You are free to use the remaining opcodes for message types specific to your implementation.

**RETURNS** N/A

**SEE ALSO** **masterIoLib**

---

## *masterIpcFree()*

**NAME** *masterIpcFree()* – free the IPC resources allocated by the SNMP master agent

**SYNOPSIS**

```
void masterIpcFree
(
 PTR_T ipchandle /* pointer to IPC handle */
)
```

**DESCRIPTION** The SNMP master agent calls this routine to free a pointer to an IPC handle. This is part of the deregistration process for an SNMP agent.

**RETURNS** N/A

**SEE ALSO** **masterIoLib**

---

## *masterIpcRcv()*

**NAME** *masterIpcRcv()* – wait for a reply from the subagent

**SYNOPSIS**

```
INT_32_T masterIpcRcv
(
 EBUFFER_T * pBuf, /* buffer to be filled */
 PTR_T ipchandle /* pointer to the IPC handle */
)
```

**DESCRIPTION** This routine waits for a response after query has been sent to the subagent. In the shipped implementation of the WindNet SNMP master agent, this function waits on a message queue that is local to the master agent. This message queue is used to facilitate communication between **tSnmpd**, the task that manages communication with the SNMP manager, and **tMonQue**, the task that manages communication between the SNMP master agent and its subagents.

In the shipped master agent code, subagents communicate with the master agent by putting messages on the message queue monitored by **tMonQue**. If the message is a control message, it is processed by *snmpMasterHandlerWR()*. If the message is a query response, it is transferred to the local message queue on which *masterIpcRcv()* is waiting. All of this is handled synchronously. Thus, while the master agent is waiting for a response from the subagent, it is blocked. Normally, the amount of time spent blocked is quite short and is not a problem.

However, it is an imperfect world, so it is possible that a response for a query never makes it back to the subagent. To handle this possibility, the shipped version of the WindNet SNMP master agent puts a timeout on its wait for a query response. If you should rewrite the SNMP master agent for any reason, make sure that you preserve this timeout.

**RETURNS** 0, if the packet was received successfully; 1, if an error or a timeout has caused the objects to be marked inactive and subsequently removed; 2, if the master agent will allow the current packet to be processed without freeing objects.

**SEE ALSO** **masterIoLib**

---

## *masterIpcSend()*

**NAME** *masterIpcSend()* – send a message to a subagent

**SYNOPSIS**

```
INT_32_T masterIpcSend
(
 EBUFFER_T * pBuf, /* message to be sent */
 PTR_T ipchandle /* address of subagent */
)
```

**DESCRIPTION** The SNMP master agent calls when it needs to send a query in *buf* to the subagent at the *ipchandle* address. If this routine is used with *snmpMasterHandlerAsync()*, you must rewrite the function according to the prototype of `IPSEND_AS_T` (see `subagent.h`). The additional parameter *reqid* in this prototype is the request ID of the message being sent. Use *reqid* to call *snmpMasterCleanup()* if the IPC layer times out.

Internally, this function calls *masterIoWrite()* to put a message on the subagent's message queue. If you have rewritten *masterIoWrite()* to use different IPC mechanism, such as sockets, you should take care that your rewrite of *masterIoWrite()* is compatible with its use in *masterIpcSend()*.

**RETURNS** 0, if the packet has been sent successfully; 1, if an error has been detected that caused the objects to be marked inactive and possibly removed; 2, if the processing of the current packet is allowed to continue without freeing up objects.

**SEE ALSO** *masterIoLib*

---

## *masterQueCleanup()*

**NAME** *masterQueCleanup()* – free resources allocated for SNMP master agent

**SYNOPSIS**

```
void masterQueCleanup (void)
```

**DESCRIPTION** This routine is called from the cleanup routine in `snmpIoLib` if the agent fails to allocate resources. This routine deletes the message queue and all other resources that have been allocated for the master agent.

**RETURNS** N/A

**SEE ALSO** *masterIoLib*

***mathHardInit()***

---

***mathHardInit()***

**NAME** *mathHardInit()* – initialize hardware floating-point math support

**SYNOPSIS** `void mathHardInit ()`

**DESCRIPTION** This routine places the addresses of the hardware high-level math functions (trigonometric functions, etc.) in a set of global variables. This allows the standard math functions (e.g., *sin()*, *pow()*) to have a single entry point but to be dispatched to the hardware or software support routines, as specified.

This routine is called from `usrConfig.c` if `INCLUDE_HW_FP` is defined. This definition causes the linker to include the floating-point hardware support library.

Certain routines in the floating-point software emulation library do not have equivalent hardware support routines. (These are primarily routines that handle single-precision floating-point numbers.) If no emulation routine address has already been put in the global variable for this function, the address of a dummy routine that logs an error message is placed in the variable; if an emulation routine address is present (the emulation initialization, via *mathSoftInit()*, must be done prior to hardware floating-point initialization), the emulation routine address is left alone. In this way, hardware routines will be used for all available functions, while emulation will be used for the missing functions.

**RETURNS** N/A

**SEE ALSO** `mathHardLib`, *mathSoftInit()*

---

***mathSoftInit()***

**NAME** *mathSoftInit()* – initialize software floating-point math support

**SYNOPSIS** `void mathSoftInit ()`

**DESCRIPTION** This routine places the addresses of the emulated high-level math functions (trigonometric functions, etc.) in a set of global variables. This allows the standard math functions (e.g., *sin()*, *pow()*) to have a single entry point but be dispatched to the hardware or software support routines, as specified.

This routine is called from `usrConfig.c` if `INCLUDE_SW_FP` is defined. This definition causes the linker to include the floating-point emulation library.

If the system is to use some combination of emulated as well as hardware coprocessor floating points, then this routine should be called before calling *mathHardInit()*.

**RETURNS** N/A

**SEE ALSO** *mathSoftLib*, *mathHardInit()*

## ***mb86940DevInit()***

**NAME** *mb86940DevInit()* – install the driver function table

**SYNOPSIS**

```
void mb86940DevInit
(
 MB86940_CHAN * pChan
)
```

**DESCRIPTION** This routine installs the driver function table. It also prevents the serial channel from functioning by disabling the interrupt.

**RETURNS** N/A

**SEE ALSO** *mb86940Sio*

## ***mb86960EndLoad()***

**NAME** *mb86960EndLoad()* – initialize the driver and device

**SYNOPSIS**

```
END_OBJ * mb86960EndLoad
(
 char * pInitString /* String to be parsed by the driver. */
)
```

**DESCRIPTION** This routine initializes the driver and puts the device to an operational state. All of the device specific parameters are passed in via the *initString*, which expects a string of the following format:

*unit:base\_addr:int\_vector:int\_level*

This routine can be called in two modes. If it is called with an empty but allocated string, it places the name of this device (that is, "fn") into the *initString* and returns 0.

If the string is allocated and not empty, the routine attempts to load the driver using the values specified in the string.

**RETURNS** An END object pointer, or NULL on error, or 0 and the name of the device if the *initString* was NULL.

**SEE ALSO** ***mb86960End***

---

## ***mb86960InitParse( )***

**NAME** ***mb86960InitParse( )*** – parse the initialization string

**SYNOPSIS**

```
STATUS mb86960InitParse
(
 MB86960_END_CTRL * pDrvCtrl, /* device pointer */
 char * pInitString /* information string */
)
```

**DESCRIPTION** Parse the input string. Fill in values in the driver control structure.

The initialization string format is:

*unit:baseAddr:ivec*

*unit*

Device unit number, a small integer. MUST always be 0.

*devBaseAddr*

Base address of the device register set

*ivec*

Interrupt vector number (used with *sysIntConnect*)

**RETURNS** OK or ERROR for invalid arguments.

**SEE ALSO** ***mb86960End***



---

## *mb86960MemInit()*

**NAME** *mb86960MemInit()* – initialize memory for the chip

**SYNOPSIS**

```
STATUS mb86960MemInit
(
 MB86960_END_CTRL * pDrvCtrl /* device to be initialized */
)
```

**DESCRIPTION** This routine is highly specific to the device.

**RETURNS** OK or ERROR.

**SEE ALSO** *mb86960End*

---

## *mb87030CtrlCreate()*

**NAME** *mb87030CtrlCreate()* – create a control structure for an MB87030 SPC

**SYNOPSIS**

```
MB_87030_SCSI_CTRL *mb87030CtrlCreate
(
 UINT8 * spcBaseAdrs, /* base address of SPC */
 int regOffset, /* addr offset between consecutive regs. */
 UINT clkPeriod, /* period of controller clock (nsec) */
 int spcDataParity, /* type of input to SPC DP (data parity) */
 FUNCPTR spcDMABytesIn, /* SCSI DMA input function */
 FUNCPTR spcDMABytesOut /* SCSI DMA output function */
)
```

**DESCRIPTION** This routine creates a data structure that must exist before the SPC chip can be used. This routine should be called once and only once for a specified SPC. It should be the first routine called, since it allocates memory for a structure needed by all other routines in the library.

After calling this routine, at least one call to *mb87030CtrlInit()* should be made before any SCSI transaction is initiated using the SPC chip.

A detailed description of the input parameters follows:

*spcBaseAdrs*

the address at which the CPU would access the lowest register of the SPC.

*regOffset*

the address offset (bytes) to access consecutive registers. (This must be a power of 2, for example, 1, 2, 4, etc.)

*clkPeriod*

the period in nanoseconds of the signal to the SPC clock input (only used for select command timeouts).

*spcDataParity*

the parity bit must be defined by one of the following constants, according to whether the input to SPC DP is GND, +5V, or a valid parity signal, respectively:

```
SPC_DATA_PARITY_LOW
SPC_DATA_PARITY_HIGH
SPC_DATA_PARITY_VALID
```

*spcDmaBytesIn* and *spcDmaBytesOut*

pointers to board-specific routines to handle DMA input and output. If these are NULL (0), SPC program transfer mode is used. DMA is possible only during SCSI data in/out phases. The interface to these DMA routines must be of the form:

```
STATUS xxDmaBytes{In, Out}
(
 SCSI_PHYS_DEV *pScsiPhysDev, /* ptr to phys dev info */
 UINT8 *pBuffer, /* ptr to the data buffer */
 int bufLength /* number of bytes to xfer */
)
```

**RETURNS** A pointer to the SPC control structure, or NULL if memory is insufficient or parameters are invalid.

**SEE ALSO** **mb87030Lib**

---

## ***mb87030CtrlInit()***

**NAME** *mb87030CtrlInit()* – initialize a control structure for an MB87030 SPC

**SYNOPSIS**

```
STATUS mb87030CtrlInit
(
 MB_87030_SCSI_CTRL * pSpc, /* ptr to SPC struct */
 int scsiCtrlBusId, /* SCSI bus ID of this SPC */
 UINT defaultSelTimeout, /* default dev sel timeout */
 int scsiPriority /* priority of task doing SCSI */
)
```

**DESCRIPTION** This routine initializes an SPC control structure created by *mb87030CtrlCreate()*. It must be called before the SPC is used. This routine can be called more than once; however, it should be called only while there is no activity on the SCSI interface.

Before returning, this routine pulses RST (reset) on the SCSI bus, thus resetting all attached devices.

The input parameters are as follows:

*pSpc*  
a pointer to the **MB\_87030\_SCSI\_CTRL** structure created with *mb87030CtrlCreate()*.

*scsiCtrlBusId*  
the SCSI bus ID of the SIOP, in the range 0 – 7. The ID is somewhat arbitrary; the value 7, or highest priority, is conventional.

*defaultSelTimeOut*  
the timeout, in microseconds, for selecting a SCSI device attached to this controller. The recommended value 0 specifies **SCSI\_DEF\_SELECT\_TIMEOUT** (250 milliseconds). The maximum timeout possible is approximately 3 seconds. Values exceeding this revert to the maximum.

*scsiPriority*  
the priority to which a task is set when performing a SCSI transaction. Valid priorities range from 0 to 255. Alternatively, the value -1 specifies that the priority should not be altered during SCSI transactions.

**RETURNS** OK, or ERROR if parameters are out of range.

**SEE ALSO** **mb87030Lib**

---

## ***mb87030Show()***

**NAME** *mb87030Show()* – display the values of all readable MB87030 SPC registers

**SYNOPSIS**

```
STATUS mb87030Show
(
 SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */
)
```

**DESCRIPTION** This routine displays the state of the SPC registers in a user-friendly manner. It is useful primarily for debugging.

**EXAMPLE**

```
-> mb87030Show
SCSI Bus ID: 7
```

***mbcAddrFilterSet()***

```

SCTL (0x01): intsEnbl
SCMD (0x00): busRlease
TMOD (0x00): asyncMode
INTS (0x00):
PSNS (0x00): req0 ack0 atn0 sel0 bsy0 msg0 c_d0 i_o0
SSTS (0x05): noConIdle xferCnt=0 dregEmpty
SERR (0x00): noParErr
PCTL (0x00): bfIntDsbl phDataOut
MBC (0x00): 0
XFER COUNT : 0x000000 = 0

```

**RETURNS** OK, or ERROR if *pScsiCtrl* and *pSysScsiCtrl* are both NULL.

**SEE ALSO** **mb87030Lib**

***mbcAddrFilterSet()***

**NAME** *mbcAddrFilterSet()* – set the address filter for multicast addresses

**SYNOPSIS**

```

void mbcAddrFilterSet
(
 MBC_DEVICE * pDrvCtrl /* device to be updated */
)

```

**DESCRIPTION** This routine goes through all of the multicast addresses on the list of addresses (added with the *endAddrAdd()* routine) and sets the device's filter correctly.

**RETURNS** N/A.

**SEE ALSO** **mbcEnd**

***mbcattach()***

**NAME** *mbcattach()* – publish the **mbc** network interface and initialize the driver

**SYNOPSIS**

```

STATUS mbcattach
(
 int unit, /* unit number */
 void * pEmBase, /* ethernet module base address */

```

```

int inum, /* interrupt vector number */
int txBdNum, /* number of transmit buffer descriptors */
int rxBdNum, /* number of receive buffer descriptors */
int dmaParms, /* DMA parameters */
UINT8 * bufBase /* address of memory pool; NONE = malloc it */
)

```

**DESCRIPTION** The routine publishes the **mbc** interface by adding an **mbc** Interface Data Record (IDR) to the global network interface list.

The Ethernet controller uses buffer descriptors from an on-chip dual-ported RAM region, while the buffers are allocated in RAM external to the controller. The buffer memory pool can be allocated in a non-cacheable RAM region and passed as parameter *bufBase*.

Otherwise *bufBase* is NULL and the buffer memory pool is allocated by the routine using *cacheDmaMalloc()*. The driver uses this buffer pool to allocate the specified number of 1518-byte buffers for transmit, receive, and loaner pools.

The parameters *txBdNum* and *rxBdNum* specify the number of buffers to allocate for transmit and receive. If either of these parameters is NULL, the default value of 2 is used. The number of loaner buffers allocated is the lesser of *rxBdNum* and 16.

The on-chip dual ported RAM can only be partitioned so that the maximum receive and maximum transmit BDs are:

- Transmit BDs: 8, Receive BDs: 120
- Transmit BDs: 16, Receive BDs: 112
- Transmit BDs: 32, Receive BDs: 96
- Transmit BDs: 64, Receive BDs: 64

**RETURNS** ERROR, if *unit* is out of range or non-cacheable memory cannot be allocated; otherwise TRUE.

**SEE ALSO** *if\_mbc*, *ifLib*, *Motorola MC68EN302 User's Manual*

---

## ***mbcEndLoad()***

**NAME** *mbcEndLoad()* – initialize the driver and device

**SYNOPSIS**

```

END_OBJ* mbcEndLoad
(
 char * initString /* String to be parsed by the driver. */
)

```

***mbcIntr()***

|                    |                                                                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the <code>initString</code> .<br><br>The string contains the target specific parameters like this:<br><br>"unit:memAddr:ivec:txBdNum:rxBdNum:dmaParms:bufBase:offset" |
| <b>RETURNS</b>     | An END object pointer or NULL on error.                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | <b><i>mbcEnd</i></b>                                                                                                                                                                                                                                                                                   |

---

***mbcIntr()***

|                    |                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b><i>mbcIntr()</i></b> – network interface interrupt handler                                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>void mbcIntr (     int unit /* unit number */ )</pre>                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | This routine is called at interrupt level. It handles work that requires minimal processing. Interrupt processing that is more extensive gets handled at task level. The network task, <i>netTask()</i> , is provided for this function. Routines get added to the <i>netTask()</i> work queue via the <i>netJobAdd()</i> command. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | <b><i>if_mbc</i></b>                                                                                                                                                                                                                                                                                                               |

---

***mbcMemInit()***

|                    |                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b><i>mbcMemInit()</i></b> – initialize memory for the chip                               |
| <b>SYNOPSIS</b>    | <pre>STATUS mbcMemInit (     MBC_DEVICE * pDrvCtrl /* device to be initialized */ )</pre> |
| <b>DESCRIPTION</b> | Allocates and initializes the memory pools for the mbc device.                            |

**RETURNS** OK or ERROR.

**SEE ALSO** **mbcEnd**

---

## *mbcParse()*

**NAME** *mbcParse()* – parse the init string

**SYNOPSIS**

```
STATUS mbcParse
(
 MBC_DEVICE * pDrvCtrl, /* device pointer */
 char * initString /* information string */
)
```

**DESCRIPTION** Parse the input string. Fill in values in the driver control structure.

The initialization string format is:

*unit:memAddr:ivec:txBdNum:rxBdNum:dmaParms:bufBase:offset*

*unit*

Device unit number, a small integer.

*memAddr*

ethernet module base address.

*ivec*

Interrupt vector number (used with sysIntConnect)

*txBdNum*

transmit buffer descriptor

*rxBdNum*

receive buffer descriptor

*dmaParms*

dma parameters

*bufBase*

address of memory pool

*offset*

packet data offset

**RETURNS** OK or ERROR for invalid arguments.

**SEE ALSO** **mbcEnd**

## ***mbcStartOutput()***

**NAME** *mbcStartOutput()* – output packet to network interface device

**SYNOPSIS**

```
#ifdef BSD43_DRIVER LOCAL void mbcStartOutput
(
 int unit /* unit number */
)
```

**DESCRIPTION** *mbcStartOutput()* takes a packet from the network interface output queue, copies the mbuf chain into an interface buffer, and sends the packet over the interface. etherOutputHookRtns are supported.

Collision stats are collected in this routine from previously sent BDs. These BDs will not be examined until after the transmitter has cycled the ring, coming upon the BD after it has been sent. Thus, collision stat collection will be delayed a full cycle through the Tx ring.

This routine is called under several possible scenarios. Each one will be described below.

The first, and most common, is when a user task requests the transmission of data. Under BSD 4.3, this results in a call to *mbcOutput()*, which in turn calls *ether\_output()*. The routine, *ether\_output()*, will make a call to *mbcStartOutput()* if our interface output queue is not full, otherwise, the outgoing data is discarded. BSD 4.4 uses a slightly different model, in which the generic *ether\_output()* routine is called directly, followed by a call to this routine.

The second scenario is when this routine, while executing runs out of free Tx BDs, turns on transmit interrupts and exits. When the next BD is transmitted, an interrupt occurs and the ISR does a *netJobAdd* of the routine which executes in the context of *netTask()* and continues sending packets from the interface output queue.

The third scenario is when the device is reset, typically when the promiscuous mode is altered; which results in a call to *mbcInit()*. This resets the device, does a *netJobAdd()* of this routine to enable transmitting queued packets.

**RETURNS** N/A

**SEE ALSO** *if\_mbc*



---

## ***mblen()***

|                      |                                                                                       |
|----------------------|---------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>mblen()</i> – calculate the length of a multibyte character (Unimplemented) (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int mblen (     const char * s,     size_t      n )</pre>                        |
| <b>DESCRIPTION</b>   | This multibyte character function is unimplemented in VxWorks.                        |
| <b>INCLUDE FILES</b> | <b>stdlib.h</b>                                                                       |
| <b>RETURNS</b>       | OK, or ERROR if the parameters are invalid.                                           |
| <b>SEE ALSO</b>      | <b>ansiStdlib</b>                                                                     |

---

## ***mbstowcs()***

|                      |                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>mbstowcs()</i> – convert a series of multibyte char's to wide char's (Unimplemented) (ANSI) |
| <b>SYNOPSIS</b>      | <pre>size_t mbstowcs (     wchar_t *  pwcs,     const char * s,     size_t      n )</pre>      |
| <b>DESCRIPTION</b>   | This multibyte character function is unimplemented in VxWorks.                                 |
| <b>INCLUDE FILES</b> | <b>stdlib.h</b>                                                                                |
| <b>RETURNS</b>       | OK, or ERROR if the parameters are invalid.                                                    |
| <b>SEE ALSO</b>      | <b>ansiStdlib</b>                                                                              |

---

## ***mbtowc()***

|                      |                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>mbtowc()</i> – convert a multibyte character to a wide character (Unimplemented) (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int mbtowc (     wchar_t *    pwc,     const char * s,     size_t      n )</pre>      |
| <b>DESCRIPTION</b>   | This multibyte character function is unimplemented in VxWorks.                             |
| <b>INCLUDE FILES</b> | <code>stdlib.h</code>                                                                      |
| <b>RETURNS</b>       | OK, or ERROR if the parameters are invalid.                                                |
| <b>SEE ALSO</b>      | <code>ansiStdlib</code>                                                                    |

---

## ***mbufShow()***

|                    |                                                                 |
|--------------------|-----------------------------------------------------------------|
| <b>NAME</b>        | <i>mbufShow()</i> – report mbuf statistics                      |
| <b>SYNOPSIS</b>    | <code>void mbufShow (void)</code>                               |
| <b>DESCRIPTION</b> | This routine displays the distribution of mbufs in the network. |
| <b>RETURNS</b>     | N/A                                                             |
| <b>SEE ALSO</b>    | <code>netShow</code>                                            |

---

## *memAddToPool()*

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memAddToPool()</i> – add memory to the system memory partition                                                                  |
| <b>SYNOPSIS</b>    | <pre>void memAddToPool (     char *   pPool, /* pointer to memory block */     unsigned poolSize /* block size in bytes */ )</pre> |
| <b>DESCRIPTION</b> | This routine adds memory to the system memory partition, after the initial allocation of memory to the system memory partition.    |
| <b>RETURNS</b>     | N/A                                                                                                                                |
| <b>SEE ALSO</b>    | <i>memPartLib</i> , <i>memPartAddToPool()</i>                                                                                      |

---

## *memalign()*

|                    |                                                                                                                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memalign()</i> – allocate aligned memory                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>void *memalign (     unsigned alignment, /* boundary to align to (power of 2) */     unsigned size       /* number of bytes to allocate */ )</pre>                                                                                                                  |
| <b>DESCRIPTION</b> | This routine allocates a buffer of size <i>size</i> from the system memory partition. Additionally, it insures that the allocated buffer begins on a memory address evenly divisible by the specified alignment parameter. The alignment parameter must be a power of 2. |
| <b>RETURNS</b>     | A pointer to the newly allocated block, or NULL if the buffer could not be allocated.                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <i>memLib</i>                                                                                                                                                                                                                                                            |

## *memchr()*

|                      |                                                                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>memchr()</i> – search a block of memory for a character (ANSI)                                                                                                                                    |
| <b>SYNOPSIS</b>      | <pre>void * memchr (     const void * m, /* block of memory */     int         c, /* character to search for */     size_t      n /* size of memory to search */ )</pre>                             |
| <b>DESCRIPTION</b>   | This routine searches for the first element of an array of <b>unsigned char</b> , beginning at the address <i>m</i> with size <i>n</i> , that equals <i>c</i> converted to an <b>unsigned char</b> . |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                                                      |
| <b>RETURNS</b>       | If successful, it returns the address of the matching element; otherwise, a null pointer.                                                                                                            |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                                                    |

---

## *memcmp()*

|                      |                                                                                                                                                                                                                           |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>memcmp()</i> – compare two blocks of memory (ANSI)                                                                                                                                                                     |
| <b>SYNOPSIS</b>      | <pre>int memcmp (     const void * s1, /* array 1 */     const void * s2, /* array 2 */     size_t      n /* size of memory to compare */ )</pre>                                                                         |
| <b>DESCRIPTION</b>   | This routine compares successive elements from two arrays of <b>unsigned char</b> , beginning at the addresses <i>s1</i> and <i>s2</i> (both of size <i>n</i> ), until it finds elements that are not equal.              |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                                                                           |
| <b>RETURNS</b>       | If all elements are equal, zero. If elements differ and the differing element from <i>s1</i> is greater than the element from <i>s2</i> , the routine returns a positive number; otherwise, it returns a negative number. |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                                                                         |

---

## *memcpy()*

|                      |                                                                                                                                                                                                                           |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>memcpy()</i> – copy memory from one location to another (ANSI)                                                                                                                                                         |
| <b>SYNOPSIS</b>      | <pre>void * memcpy (     void *      destination, /* destination of copy */     const void * source,    /* source of copy */     size_t     size        /* size of memory to copy */ )</pre>                              |
| <b>DESCRIPTION</b>   | This routine copies <i>size</i> characters from the object pointed to by <i>source</i> into the object pointed to by <i>destination</i> . If copying takes place between objects that overlap, the behavior is undefined. |
| <b>INCLUDE FILES</b> | <code>string.h</code>                                                                                                                                                                                                     |
| <b>RETURNS</b>       | A pointer to <i>destination</i> .                                                                                                                                                                                         |
| <b>SEE ALSO</b>      | <code>ansiString</code>                                                                                                                                                                                                   |

---

## *memDevCreate()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memDevCreate()</i> – create a memory device                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>STATUS memDevCreate (     char * name, /* device name */     char * base, /* where to start in memory */     int   length /* number of bytes */ )</pre>                                                                                                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine creates a memory device containing a single file. Memory for the device is simply an absolute memory location beginning at <i>base</i>. The <i>length</i> parameter indicates the size of memory.</p> <p>For example, to create the device <code>"/mem/cpu0/"</code>, a device for accessing the entire memory of the local processor, the proper call would be:</p> <pre>memDevCreate ("/mem/cpu0/", 0, sysMemTop())</pre> <p>The device is created with the specified name, start location, and size.</p> |

To open a file descriptor to the memory, use *open()*. Specify a pseudo-file name of the byte offset desired, or open the "raw" file at the beginning and specify a position to seek to. For example, the following call to *open()* allows memory to be read starting at decimal offset 1000.

```
-> fd = open ("/mem/cpu0/1000", O_RDONLY, 0)
```

Pseudo-file name offsets are scanned with "%d".

**CAVEAT** The FIOSEEK operation overrides the offset given via the pseudo-file name at open time.

**EXAMPLE** Consider a system configured with two CPUs in the backplane and a separate dual-ported memory board, each with 1 megabyte of memory. The first CPU is mapped at VMEbus address 0x00400000 (4 Meg.), the second at bus address 0x00800000 (8 Meg.), the dual-ported memory board at 0x00c00000 (12 Meg.). Three devices can be created on each CPU as follows. On processor 0:

```
-> memDevCreate ("/mem/local/", 0, sysMemTop())
...
-> memDevCreate ("/mem/cpu1/", 0x00800000, 0x00100000)
...
-> memDevCreate ("/mem/share/", 0x00c00000, 0x00100000)
```

On processor 1:

```
-> memDevCreate ("/mem/local/", 0, sysMemTop())
...
-> memDevCreate ("/mem/cpu0/", 0x00400000, 0x00100000)
...
-> memDevCreate ("/mem/share/", 0x00c00000, 0x00100000)
```

Processor 0 has a local disk. Data or an object module needs to be passed from processor 0 to processor 1. To accomplish this, processor 0 first calls:

```
-> copy </disk1/module.o >/mem/share/0
```

Processor 1 can then be given the load command:

```
-> ld </mem/share/0
```

**RETURNS** OK, or ERROR if memory is insufficient or the I/O system cannot add the device.

**ERRNO** S\_ioLib\_NO\_DRIVER

**SEE ALSO** memDrv

---

## *memDevCreateDir()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memDevCreateDir()</i> – create a memory device for multiple files                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>STATUS memDevCreateDir (     char *          name,      /* device name */     MEM_DRV_DIRENTRY * files, /* array of dir. entries - not copied */     int            numFiles /* number of entries */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This routine creates a memory device for a collection of files organised into directories. The given array of directory entry records describes a number of files, some of which may be directories, represented by their own directory entry arrays. The structure may be arbitrarily deep. This effectively allows a filesystem to be created and installed in VxWorks, for essentially read-only use. The filesystem structure can be created on the host using the memdrvbuild utility.</p> <p>Note that the array supplied is not copied; a reference to it is kept. This array should not be modified after being passed to memDevCreateDir.</p> |
| <b>RETURNS</b>     | OK, or ERROR if memory is insufficient or the I/O system cannot add the device.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>ERRNO</b>       | S_ioLib_NO_DRIVER                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | memDrv                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

---

## *memDevDelete()*

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memDevDelete()</i> – delete a memory device                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS memDevDelete (     char * name /* device name */ )</pre>                                                                                                                                                                                                      |
| <b>DESCRIPTION</b> | <p>This routine deletes a memory device containing a single file or a collection of files. The device is deleted with its own name.</p> <p>For example, to delete the device created by memDevCreate ("/mem/cpu0/", 0, <i>sysMemTop()</i>), the proper call would be:</p> |

```
memDevDelete ("/mem/cpu0/");
```

**RETURNS** OK, or ERROR if the device doesn't exist.

**SEE ALSO** **memDrv**

---

## ***memDrv()***

**NAME** *memDrv()* – install a memory driver

**SYNOPSIS** `STATUS memDrv (void)`

**DESCRIPTION** This routine initializes the memory driver. It must be called first, before any other routine in the driver.

**RETURNS** OK, or ERROR if the I/O system cannot install the driver.

**SEE ALSO** **memDrv**

---

## ***memFindMax()***

**NAME** *memFindMax()* – find the largest free block in the system memory partition

**SYNOPSIS** `int memFindMax (void)`

**DESCRIPTION** This routine searches for the largest block in the system memory partition free list and returns its size.

**RETURNS** The size, in bytes, of the largest available block.

**SEE ALSO** **memLib**, **memPartFindMax()**



---

## *memmove()*

|                      |                                                                                                                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>memmove()</i> – copy memory from one location to another (ANSI)                                                                                                                                                       |
| <b>SYNOPSIS</b>      | <pre>void * memmove (     void *      destination, /* destination of copy */     const void * source,    /* source of copy */     size_t      size        /* size of memory to copy */ )</pre>                           |
| <b>DESCRIPTION</b>   | This routine copies <i>size</i> characters from the memory location <i>source</i> to the location <i>destination</i> . It ensures that the memory is not corrupted even if <i>source</i> and <i>destination</i> overlap. |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                                                                          |
| <b>RETURNS</b>       | A pointer to <i>destination</i> .                                                                                                                                                                                        |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                                                                        |

---

## *memOptionsSet()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memOptionsSet()</i> – set the debug options for the system memory partition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>void memOptionsSet (     unsigned options /* options for system partition */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b> | <p>This routine sets the debug options for the system memory partition. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, the following options can be selected for actions to be taken when the error is detected: (1) return the error status, (2) log an error message and return the error status, or (3) log an error message and suspend the calling task.</p> <p>These options are discussed in detail in the library manual entry for <b>memLib</b>.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <b>memLib</b> , <i>memPartOptionsSet()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

## ***memPartAddToPool()***

**NAME** *memPartAddToPool()* – add memory to a memory partition

**SYNOPSIS**

```
STATUS memPartAddToPool
(
 PART_ID partId, /* partition to initialize */
 char * pPool, /* pointer to memory block */
 unsigned poolSize /* block size in bytes */
)
```

**DESCRIPTION** This routine adds memory to a specified memory partition already created with *memPartCreate()*. The memory added need not be contiguous with memory previously assigned to the partition.

**RETURNS** OK or ERROR.

**ERRNO** S\_smObjLib\_NOT\_INITIALIZED, S\_memLib\_INVALID\_NBYTES

**SEE ALSO** *memPartLib*, *smMemLib*, *memPartCreate()*

---

## ***memPartAlignedAlloc()***

**NAME** *memPartAlignedAlloc()* – allocate aligned memory from a partition

**SYNOPSIS**

```
void *memPartAlignedAlloc
(
 PART_ID partId, /* memory partition to allocate from */
 unsigned nBytes, /* number of bytes to allocate */
 unsigned alignment /* boundary to align to */
)
```

**DESCRIPTION** This routine allocates a buffer of size *nBytes* from a specified partition. Additionally, it insures that the allocated buffer begins on a memory address evenly divisible by *alignment*. The *alignment* parameter must be a power of 2.

**RETURNS** A pointer to the newly allocated block, or NULL if the buffer could not be allocated.

**SEE ALSO** *memPartLib*

---

## *memPartAlloc()*

|                    |                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memPartAlloc()</i> – allocate a block of memory from a partition                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>void *memPartAlloc (     PART_ID partId, /* memory partition to allocate from */     unsigned nBytes /* number of bytes to allocate */ )</pre>                                                             |
| <b>DESCRIPTION</b> | This routine allocates a block of memory from a specified partition. The size of the block will be equal to or greater than <i>nBytes</i> . The partition must already be created with <i>memPartCreate()</i> . |
| <b>RETURNS</b>     | A pointer to a block, or NULL if the call fails.                                                                                                                                                                |
| <b>ERRNO</b>       | S_smObjLib_NOT_INITIALIZED                                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <i>memPartLib</i> , <i>smMemLib</i> , <i>memPartCreate()</i>                                                                                                                                                    |

---

## *memPartCreate()*

|                    |                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memPartCreate()</i> – create a memory partition                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>PART_ID memPartCreate (     char * pPool, /* pointer to memory area */     unsigned poolSize /* size in bytes */ )</pre>                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | This routine creates a new memory partition containing a specified memory pool. It returns a partition ID, which can then be passed to other routines to manage the partition (i.e., to allocate and free memory blocks in the partition). Partitions can be created to manage any number of separate memory pools. |
| <b>NOTE</b>        | The descriptor for the new partition is allocated out of the system memory partition (i.e., with <i>malloc()</i> ).                                                                                                                                                                                                 |
| <b>RETURNS</b>     | The partition ID, or NULL if there is insufficient memory in the system memory partition for a new partition descriptor.                                                                                                                                                                                            |

SEE ALSO **memPartLib, smMemLib**

---

## ***memPartFindMax()***

**NAME** *memPartFindMax()* – find the size of the largest available free block

**SYNOPSIS**

```
int memPartFindMax
(
 PART_ID partId /* partition ID */
)
```

**DESCRIPTION** This routine searches for the largest block in the memory partition free list and returns its size.

**RETURNS** The size, in bytes, of the largest available block.

**ERRNO** S\_smObjLib\_NOT\_INITIALIZED

**SEE ALSO** **memLib, smMemLib**

---

## ***memPartFree()***

**NAME** *memPartFree()* – free a block of memory in a partition

**SYNOPSIS**

```
STATUS memPartFree
(
 PART_ID partId, /* memory partition to add block to */
 char * pBlock /* pointer to block of memory to free */
)
```

**DESCRIPTION** This routine returns to a partition's free memory list a block of memory previously allocated with *memPartAlloc()*.

**RETURNS** OK, or ERROR if the block is invalid.

**ERRNO** S\_smObjLib\_NOT\_INITIALIZED

**SEE ALSO** **memPartLib, smMemLib, memPartAlloc()**

---

## *memPartInfoGet()*

|                    |                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memPartInfoGet()</i> – get partition information                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>STATUS memPartInfoGet (     PART_ID      partId,    /* partition ID */     MEM_PART_STATS * ppartStats /* partition stats structure */ )</pre>                      |
| <b>DESCRIPTION</b> | This routine takes a partition ID and a pointer to a MEM_PART_STATS structure. All the parameters of the structure are filled in with the current partition information. |
| <b>RETURNS</b>     | OK if the structure has valid data, otherwise ERROR.                                                                                                                     |
| <b>SEE ALSO</b>    | <i>memShow()</i>                                                                                                                                                         |

---

## *memPartOptionsSet()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memPartOptionsSet()</i> – set the debug options for a memory partition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS memPartOptionsSet (     PART_ID partId, /* partition to set option for */     unsigned options /* memory management options */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | <p>This routine sets the debug options for a specified memory partition. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, the error status is returned. There are four error-handling options that can be individually selected:</p> <p><b>MEM_ALLOC_ERROR_LOG_FLAG</b><br/>Log a message when there is an error in allocating memory.</p> <p><b>MEM_ALLOC_ERROR_SUSPEND_FLAG</b><br/>Suspend the task when there is an error in allocating memory (unless the task was spawned with the VX_UNBREAKABLE option, in which case it cannot be suspended).</p> <p><b>MEM_BLOCK_ERROR_LOG_FLAG</b><br/>Log a message when there is an error in freeing memory.</p> |

**MEM\_BLOCK\_ERROR\_SUSPEND\_FLAG**

Suspend the task when there is an error in freeing memory (unless the task was spawned with the **VX\_UNBREAKABLE** option, in which case it cannot be suspended).

These options are discussed in detail in the library manual entry for **memLib**.

**RETURNS** OK or ERROR.  
**ERRNO** S\_smObjLib\_NOT\_INITIALIZED  
**SEE ALSO** memLib, smMemLib

---

## *memPartRealloc()*

**NAME** *memPartRealloc()* – reallocate a block of memory in a specified partition

**SYNOPSIS**

```
void *memPartRealloc
(
 PART_ID partId, /* partition ID */
 char * pBlock, /* block to be reallocated */
 unsigned nBytes /* new block size in bytes */
)
```

**DESCRIPTION** This routine changes the size of a specified block of memory and returns a pointer to the new block. The contents that fit inside the new size (or old size if smaller) remain unchanged. The memory alignment of the new block is not guaranteed to be the same as the original block.

If *pBlock* is NULL, this call is equivalent to *memPartAlloc()*.

**RETURNS** A pointer to the new block of memory, or NULL if the call fails.  
**ERRNO** S\_smObjLib\_NOT\_INITIALIZED  
**SEE ALSO** memLib, smMemLib

---

## *memPartShow()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memPartShow()</i> – show partition blocks and statistics                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS memPartShow (     PART_ID partId, /* partition ID */     int     type    /* 0 = statistics, 1 = statistics &amp; list */ )</pre>                                                                                                                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This routine displays statistics about the available and allocated memory in a specified memory partition. It shows the number of bytes, the number of blocks, and the average block size in both free and allocated memory, and also the maximum block size of free memory. It also shows the number of blocks currently allocated and the average allocated block size.</p> <p>In addition, if <i>type</i> is 1, the routine displays a list of all the blocks in the free list of the specified partition.</p> |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>ERRNO</b>       | S_smObjLib_NOT_INITIALIZED                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>memShow()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                                                                                                                                                                                                              |

---

## *memPartSmCreate()*

|                    |                                                                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memPartSmCreate()</i> – create a shared memory partition (VxMP Opt.)                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>PART_ID memPartSmCreate (     char *  pPool, /* global address of shared memory area */     unsigned poolSize /* size in bytes */ )</pre>                                                                                                                                           |
| <b>DESCRIPTION</b> | <p>This routine creates a shared memory partition that can be used by tasks on all CPUs in the system. It returns a partition ID which can then be passed to generic <b>memPartLib</b> routines to manage the partition (i.e., to allocate and free memory blocks in the partition).</p> |

**memset()**

*pPool* is the global address of shared memory dedicated to the partition. The memory area pointed to by *pPool* must be in the same address space as the shared memory anchor and shared memory pool.

*poolSize* is the size in bytes of shared memory dedicated to the partition.

Before this routine can be called, the shared memory objects facility must be initialized (see **smMemLib**).

|                     |                                                                                                                                                                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NOTE</b>         | The descriptor for the new partition is allocated out of an internal dedicated shared memory partition. The maximum number of partitions that can be created is <b>SM_OBJ_MAX_MEM_PART</b> .<br><br>Memory pool size is rounded down to a 16-byte boundary. |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.                                                                                                                                                     |
| <b>RETURNS</b>      | The partition ID, or NULL if there is insufficient memory in the dedicated partition for a new partition descriptor.                                                                                                                                        |
| <b>ERRNO</b>        | <b>S_memLib_NOT_ENOUGH_MEMORY</b><br><b>S_smObjLib_LOCK_TIMEOUT</b>                                                                                                                                                                                         |
| <b>SEE ALSO</b>     | <b>smMemLib</b> , <b>memLib</b>                                                                                                                                                                                                                             |

---

**memset()**

|                      |                                                                                                                                                                                |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>memset()</b> – set a block of memory (ANSI)                                                                                                                                 |
| <b>SYNOPSIS</b>      | <pre>void * memset (     void * m,    /* block of memory */     int  c,     /* character to store */     size_t size /* size of memory */ )</pre>                              |
| <b>DESCRIPTION</b>   | This routine stores <i>c</i> converted to an <b>unsigned char</b> in each of the elements of the array of <b>unsigned char</b> beginning at <i>m</i> , with size <i>size</i> . |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                                |
| <b>RETURNS</b>       | A pointer to <i>m</i> .                                                                                                                                                        |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                              |



---

## *memShow()*

**NAME** *memShow()* – show system memory partition blocks and statistics

**SYNOPSIS**

```
void memShow
(
 int type /* 1 = list all blocks in the free list */
)
```

**DESCRIPTION** This routine displays statistics about the available and allocated memory in the system memory partition. It shows the number of bytes, the number of blocks, and the average block size in both free and allocated memory, and also the maximum block size of free memory. It also shows the number of blocks currently allocated and the average allocated block size.

In addition, if *type* is 1, the routine displays a list of all the blocks in the free list of the system partition.

**EXAMPLE**

```
-> memShow 1
FREE LIST:
 num addr size
 ---- -
 1 0x3fee18 16
 2 0x3b1434 20
 3 0x4d188 2909400
SUMMARY:
status bytes blocks avg block max block

current
 free 2909436 3 969812 2909400
 alloc 969060 16102 60 -
cumulative
 alloc 1143340 16365 69 -
```

**RETURNS** N/A

**SEE ALSO** *memShow*, *memPartShow()*, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## *memShowInit()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>memShowInit()</i> – initialize the memory partition show facility                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <code>void memShowInit (void)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine links the memory partition show facility into the VxWorks system. These routines are included automatically when this show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none"><li>– If you use the configuration header files, define <code>INCLUDE_SHOW_ROUTINES</code> in <code>config.h</code>.</li><li>– If you use the Tornado project facility, select <code>INCLUDE_MEM_SHOW</code>.</li></ul> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <code>memShow</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

---

## *mib2ErrorAdd()*

|                    |                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>mib2ErrorAdd()</i> – change a MIB-II error count                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS mib2ErrorAdd (     M2_INTERFACETBL * pMib,     int               errCode,     int               value )</pre>                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This function adds a specified value to one of the MIB-II error counters in a MIB-II interface table. The counter to be altered is specified by the <code>errCode</code> argument. Specifying a negative value reduces the error count, a positive value increases the error count.</p> |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <code>endLib</code>                                                                                                                                                                                                                                                                        |

---

## *mib2Init()*

**NAME** *mib2Init()* – initialize a MIB-II structure

**SYNOPSIS**

```
STATUS mib2Init
(
 M2_INTERFACETBL * pMib, /* struct to be initialized */
 long ifType, /* ifType from m2Lib.h */
 UCHAR * phyAddr, /* MAC/PHY address */
 int addrLength, /* MAC/PHY address length */
 int mtuSize, /* MTU size */
 int speed /* interface speed */
)
```

**DESCRIPTION** Initialize a MIB-II structure. Set all error counts to zero. Assume a 10Mbps Ethernet device.

**RETURNS** OK or ERROR.

**SEE ALSO** *endLib*

---

## *mkdir()*

**NAME** *mkdir()* – make a directory

**SYNOPSIS**

```
STATUS mkdir
(
 char * dirName /* directory name */
)
```

**DESCRIPTION** This command creates a new directory in a hierarchical file system. The *dirName* string specifies the name to be used for the new directory, and can be either a full or relative pathname.

This call is supported by the VxWorks NFS and dosFs file systems.

**RETURNS** OK, or ERROR if the directory cannot be created.

**SEE ALSO** *usrLib, rmdir()*, *VxWorks Programmer's Guide: Target Shell*

***mktime()***

---

***mktime()***

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>mktime()</i> – convert broken-down time into calendar time (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>      | <pre>time_t mktime (     struct tm * timeptr /* pointer to broken-down structure */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>   | <p>This routine converts the broken-down time, expressed as local time, in the structure pointed to by <i>timeptr</i> into a calendar time value with the same encoding as that of the values returned by the <i>time()</i> function. The original values of the <b>tm_wday</b> and <b>tm_yday</b> components of the <b>tm</b> structure are ignored, and the original values of the other components are not restricted to the ranges indicated in <b>time.h</b>. On successful completion, the values of <b>tm_wday</b> and <b>tm_yday</b> are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to the ranges indicated in <b>time.h</b>; the final value of <b>tm_mday</b> is not set until <b>tm_mon</b> and <b>tm_year</b> are determined.</p> |
| <b>INCLUDE FILES</b> | <b>time.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>RETURNS</b>       | The calendar time in seconds, or ERROR (-1) if calendar time cannot be calculated.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>      | <b>ansiTime</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

---

***mlock()***

|                    |                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>mlock()</i> – lock specified pages into memory (POSIX)                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>int mlock (     const void * addr,     size_t      len )</pre>                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This routine guarantees that the specified pages are memory resident. In VxWorks, the <i>addr</i> and <i>len</i> arguments are ignored, since all pages are memory resident.</p> |
| <b>RETURNS</b>     | 0 (OK) always.                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <b>mmanPxLib</b>                                                                                                                                                                    |

---

## *mlockall()*

- NAME** *mlockall()* – lock all pages used by a process into memory (POSIX)
- SYNOPSIS**
- ```
int mlockall
(
    int flags
)
```
- DESCRIPTION** This routine guarantees that all pages used by a process are memory resident. In VxWorks, the *flags* argument is ignored, since all pages are memory resident.
- RETURNS** 0 (OK) always.
- SEE ALSO** *mmanPxLib*

mmuL64862DmaInit()

- NAME** *mmuL64862DmaInit()* – initialize the L64862 I/O MMU DMA data structures (SPARC)
- SYNOPSIS**
- ```
STATUS mmuL64862DmaInit
(
 void * vrtBase, /* First valid DMA virtual address */
 void * vrtTop, /* Last valid DMA virtual address */
 UINT range /* range covered by I/O Page Table */
)
```
- DESCRIPTION** This routine initializes the I/O MMU in the LSI Logic L64862 MBus to SBus Interface Chip (MS) for S-Bus DMA with the TI TMS390 SuperSPARC. It assumes *cacheLib* and *vmLib* have been initialized and that the TI TMS390 Processor MMU is enabled.
- It initializes the I/O MMU to map all valid virtual addresses  $\geq$  *vrtBase* and  $\leq$  *vrtTop*. It is usually called as follows:
- ```
(void)mmuL64862DmaInit ((void *) LOCAL_MEM_LOCAL_ADRS,
                       (void *) (LOCAL_MEM_LOCAL_ADRS + LOCAL_MEM_SIZE - 1),
                       IOMMU_IOC_R_RANGE);
```
- RETURNS** OK, or ERROR if the request cannot be satisfied.
- SEE ALSO** *mmuL64862Lib*

mmuPro32LibInit()

| | |
|-------------|---|
| NAME | <i>mmuPro32LibInit()</i> – initialize module |
| SYNOPSIS | <pre>STATUS mmuPro32LibInit (int pageSize /* system pageSize (must be 4KB or 4MB) */)</pre> |
| DESCRIPTION | Build a dummy translation table that will hold the page table entries for the global translation table. The mmu remains disabled upon completion. |
| RETURNS | OK if no error, ERROR otherwise |
| ERRNO | S_mmuLib_INVALID_PAGE_SIZE |
| SEE ALSO | <i>mmuPro32Lib</i> |

mmuSparcRomInit()

| | |
|-------------|---|
| NAME | <i>mmuSparcRomInit()</i> – initialize the MMU for the ROM (SPARC) |
| SYNOPSIS | <pre>STATUS mmuSparcRomInit (int * mmuTableAdrs, /* address for the MMU tables */ int mmuRomPhysAdrs, /* ROM physical address */ int romInitAdrs /* address where romInit was linked in */)</pre> |
| DESCRIPTION | This routine initializes the MMU when the system is booted. It should be called only from <i>romInit()</i> . This routine is necessary because MMU libraries are not initialized by the boot code in bootConfig; they are initialized only in the VxWorks image in usrConfig. The same sysPhysMemDesc is used by this routine as well as <i>usrMmuInit()</i> in usrConfig to maintain consistency. |
| RETURNS | OK. |
| SEE ALSO | <i>mmuSparcLib</i> |

modf()

| | |
|----------------------|--|
| NAME | <i>modf()</i> – separate a floating-point number into integer and fraction parts (ANSI) |
| SYNOPSIS | <pre>double modf (double value, /* value to split */ double * pIntPart /* where integer portion is stored */)</pre> |
| DESCRIPTION | This routine stores the integer portion of <i>value</i> in <i>pIntPart</i> and returns the fractional portion. Both parts are double precision and will have the same sign as <i>value</i> . |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision fractional portion of <i>value</i> . |
| SEE ALSO | ansiMath , <i>frexp()</i> , <i>ldexp()</i> |

moduleCheck()

| | |
|--------------------|---|
| NAME | <i>moduleCheck()</i> – verify checksums on all modules |
| SYNOPSIS | <pre>STATUS moduleCheck (int options /* validation options */)</pre> |
| DESCRIPTION | <p>This routine verifies the checksums on the segments of all loaded modules. If any of the checksums are incorrect, a message is printed to the console, and the routine returns ERROR.</p> <p>By default, only the text segment checksum is validated.</p> <p>Bits in the <i>options</i> parameter may be set to control specific checks:</p> <p>MODCHECK_TEXT Validate the checksum for the TEXT segment (default).</p> <p>MODCHECK_DATA Validate the checksum for the DATA segment.</p> |

MODCHECK_BSS

Validate the checksum for the BSS segment.

MODCHECK_NOPRINT

Do not print a message (*moduleCheck()* still returns ERROR on failure.)

See the definitions in **moduleLib.h**

RETURNS OK, or ERROR if the checksum is invalid.

SEE ALSO **moduleLib**

moduleCreate()

NAME *moduleCreate()* – create and initialize a module

SYNOPSIS

```
MODULE_ID moduleCreate
(
    char * name, /* module name */
    int format, /* object module format */
    int flags /* symFlag as passed to loader (see loadModuleAt()) */
)
```

DESCRIPTION This routine creates an object module descriptor.

The arguments specify the name of the object module file, the object module format, and an argument specifying which symbols to add to the symbol table. See the *loadModuleAt()* description of *symFlag* for possible *flags* values.

Space for the new module is dynamically allocated.

RETURNS **MODULE_ID**, or NULL if there is an error.

SEE ALSO **moduleLib**, *loadModuleAt()*

moduleCreateHookAdd()

NAME *moduleCreateHookAdd()* – add a routine to be called when a module is added

SYNOPSIS

```
STATUS moduleCreateHookAdd
(
    FUNCPTR moduleCreateHookRtn /* routine called when module is added */
)
```

DESCRIPTION This routine adds a specified routine to a list of routines to be called when a module is created. The specified routine should be declared as follows:

```
void moduleCreateHook
(
    MODULE_ID moduleId /* the module ID */
)
```

This routine is called after all fields of the module ID have been filled in.

NOTE Modules do not have information about their object segments when they are created. This information is not available until after the entire load process has finished.

RETURNS OK or ERROR.

SEE ALSO *moduleLib, moduleCreateHookDelete()*

moduleCreateHookDelete()

NAME *moduleCreateHookDelete()* – delete a previously added module create hook routine

SYNOPSIS

```
STATUS moduleCreateHookDelete
(
    FUNCPTR moduleCreateHookRtn /* routine called when module is added */
)
```

DESCRIPTION This routine removes a specified routine from the list of routines to be called at each *moduleCreate()* call.

RETURNS OK, or ERROR if the routine is not in the table of module create hook routines.

SEE ALSO *moduleLib, moduleCreateHookAdd()*

moduleDelete()

NAME *moduleDelete()* – delete module ID information (use *unld()* to reclaim space)

SYNOPSIS

```
STATUS moduleDelete
(
    MODULE_ID moduleId /* module to delete */
)
```

DESCRIPTION

This routine deletes a module descriptor, freeing any space that was allocated for the use of the module ID.

This routine does not free space allocated for the object module itself -- this is done by *unld()*.

RETURNS OK or ERROR.

SEE ALSO **moduleLib**

moduleFindByGroup()

NAME *moduleFindByGroup()* – find a module by group number

SYNOPSIS

```
MODULE_ID moduleFindByGroup
(
    int groupNumber /* group number to find */
)
```

DESCRIPTION

This routine searches for a module with a group number matching *groupNumber*.

RETURNS MODULE_ID, or NULL if no match is found.

SEE ALSO **moduleLib**

moduleFindByName()

NAME *moduleFindByName()* – find a module by name

SYNOPSIS

```
MODULE_ID moduleFindByName
(
    char * moduleName /* name of module to find */
)
```

DESCRIPTION This routine searches for a module with a name matching *moduleName*.

RETURNS MODULE_ID, or NULL if no match is found.

SEE ALSO **moduleLib**

moduleFindByNameAndPath()

NAME *moduleFindByNameAndPath()* – find a module by file name and path

SYNOPSIS

```
MODULE_ID moduleFindByNameAndPath
(
    char * moduleName, /* file name to find */
    char * pathName    /* path name to find */
)
```

DESCRIPTION This routine searches for a module with a name matching *moduleName* and path matching *pathName*.

RETURNS MODULE_ID, or NULL if no match is found.

SEE ALSO **moduleLib**

moduleFlagsGet()

NAME *moduleFlagsGet()* – get the flags associated with a module ID

SYNOPSIS

```
int moduleFlagsGet
(
    MODULE_ID moduleId
)
```

DESCRIPTION This routine returns the flags associated with a module ID.

RETURNS The flags associated with the module ID, or NULL if the module ID is invalid.

SEE ALSO **moduleLib**

moduleIdListGet()

NAME *moduleIdListGet()* – get a list of loaded modules

SYNOPSIS

```
int moduleIdListGet
(
    MODULE_ID * idList,    /* array of module IDs to be filled in */
    int        maxModules /* max modules idList can accommodate */
)
```

DESCRIPTION This routine provides the calling task with a list of all loaded object modules. An unsorted list of module IDs for no more than *maxModules* modules is put into *idList*.

RETURNS The number of modules put into the ID list, or ERROR.

SEE ALSO **moduleLib**

moduleInfoGet()

NAME *moduleInfoGet()* – get information about an object module

SYNOPSIS

```
STATUS moduleInfoGet
(
    MODULE_ID    moduleId, /* module to return information about */
    MODULE_INFO * pModuleInfo /* pointer to module info struct */
)
```

DESCRIPTION This routine fills in a MODULE_INFO structure with information about the specified module.

RETURNS OK or ERROR.

SEE ALSO **moduleLib**

moduleNameGet()

NAME *moduleNameGet()* – get the name associated with a module ID

SYNOPSIS

```
char * moduleNameGet
(
    MODULE_ID moduleId
)
```

DESCRIPTION This routine returns a pointer to the name associated with a module ID.

RETURNS A pointer to the module name, or NULL if the module ID is invalid.

SEE ALSO **moduleLib**

moduleSegFirst()

- NAME** *moduleSegFirst()* – find the first segment in a module
- SYNOPSIS**
- ```
SEGMENT_ID moduleSegFirst
(
 MODULE_ID moduleId /* module to get segment from */
)
```
- DESCRIPTION** This routine returns information about the first segment of a module descriptor.
- RETURNS** A pointer to the segment ID, or NULL if the segment list is empty.
- SEE ALSO** *moduleLib*, *moduleSegGet()*
- 

## *moduleSegGet()*

- NAME** *moduleSegGet()* – get (delete and return) the first segment from a module
- SYNOPSIS**
- ```
SEGMENT_ID moduleSegGet  
(  
    MODULE_ID moduleId /* module to get segment from */  
)
```
- DESCRIPTION** This routine returns information about the first segment of a module descriptor, and then deletes the segment from the module.
- RETURNS** A pointer to the segment ID, or NULL if the segment list is empty.
- SEE ALSO** *moduleLib*, *moduleSegFirst()*

moduleSegNext()

| | |
|--------------------|---|
| NAME | <i>moduleSegNext()</i> – find the next segment in a module |
| SYNOPSIS | <pre>SEGMENT_ID moduleSegNext (SEGMENT_ID segmentId /* segment whose successor is to be found */)</pre> |
| DESCRIPTION | This routine returns the segment in the list immediately following <i>segmentId</i> . |
| RETURNS | A pointer to the segment ID, or NULL if there is no next segment. |
| SEE ALSO | moduleLib |

moduleShow()

| | |
|--------------------|--|
| NAME | <i>moduleShow()</i> – show the current status for all the loaded modules |
| SYNOPSIS | <pre>STATUS moduleShow (char * moduleNameOrId, /* name or ID of the module to show */ int options /* display options */)</pre> |
| DESCRIPTION | <p>This routine displays a list of the currently loaded modules and some information about where the modules are loaded.</p> <p>The specific information displayed depends on the format of the object modules. In the case of a.out and ECOFF object modules, <i>moduleShow()</i> displays the start of the text, data, and BSS segments.</p> <p>If <i>moduleShow()</i> is called with no arguments, a summary list of all loaded modules is displayed. It can also be called with an argument, <i>moduleNameOrId</i>, which can be either the name of a loaded module or a module ID. If it is called with either of these, more information about the specified module will be displayed.</p> |
| RETURNS | OK or ERROR. |
| SEE ALSO | moduleLib , <i>VxWorks Programmer's Guide: Target Shell</i> , windsh , <i>Tornado User's Guide: Shell</i> |

motCpmEndLoad()

NAME *motCpmEndLoad()* – initialize the driver and device

SYNOPSIS

```
END_OBJ *motCpmEndLoad
(
    char * initString /* parameter string */
)
```

DESCRIPTION This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the *initString*, which is of the following format:

unit:motCpmAddr:ivec:sccNum:txBdNum:rxBdNum:txBdBase:rxBdBase:bufBase

The parameters of this string are individually described in the **motCpmEnd** man page.

The SCC shares a region of memory with the driver. The caller of this routine can specify the address of a non-cacheable memory region with *bufBase*. Or, if this parameter is "NONE", the driver obtains this memory region by making calls to *cacheDmaMalloc()*. Non-cacheable memory space is important whenever the host processor uses cache memory. This is also the case when the MC68EN360 is operating in companion mode and is attached to a processor with cache memory.

After non-cacheable memory is obtained, this routine divides up the memory between the various buffer descriptors (BDs). The number of BDs can be specified by *txBdNum* and *rxBdNum*, or if "NULL", a default value of 32 BDs will be used. An additional number of buffers are reserved as receive loaner buffers. The number of loaner buffers is a default number of 16.

The user must specify the location of the transmit and receive BDs in the processor's dual ported RAM. *txBdBase* and *rxBdBase* give the offsets from *motCpmAddr* for the base of the BD rings. Each BD uses 8 bytes. Care must be taken so that the specified locations for Ethernet BDs do not conflict with other dual ported RAM structures.

Multiple individual device units are supported by this driver. Device units can reside on different chips, or could be on different SCCs within a single processor. The *sccNum* parameter is used to explicitly state which SCC is being used. SCC1 is most commonly used, thus this parameter most often equals "1".

Before this routine returns, it connects up the interrupt vector *ivec*.

RETURNS An END object pointer or NULL on error.

SEE ALSO **motCpmEnd**, *Motorola MC68EN360 User's Manual*, *Motorola MPC860 User's Manual*, *Motorola MPC821 User's Manual*

motFecEndLoad()

| | |
|--------------------|---|
| NAME | <i>motFecEndLoad()</i> – initialize the driver and device |
| SYNOPSIS | <pre>END_OBJ* motFecEndLoad (char * initString /* parameter string */)</pre> |
| DESCRIPTION | <p>This routine initializes both driver and device to an operational state using device specific parameters specified by <i>initString</i>.</p> <p>The parameter string, <i>initString</i>, is an ordered list of parameters each separated by a colon. The format of <i>initString</i> is:</p> <pre>"motCpmAddr:ivec:bufBase:bufSize:fifoTxBase:fifoRxBase:tbdNum:rbdNum:phyAddr:isoPhyAddr: phyDefMode:userFlags"</pre> <p>The FEC shares a region of memory with the driver. The caller of this routine can specify the address of this memory region, or can specify that the driver must obtain this memory region from the system resources.</p> <p>A default number of transmit/receive buffer descriptors of 32 can be selected by passing zero in the parameters <i>tbdNum</i> and <i>rbdNum</i>. In other cases, the number of buffers selected should be greater than two.</p> <p>The <i>bufBase</i> parameter is used to inform the driver about the shared memory region. If this parameter is set to the constant "NONE," then this routine will attempt to allocate the shared memory from the system. Any other value for this parameter is interpreted by this routine as the address of the shared memory region to be used. The <i>bufSize</i> parameter is used to check that this region is large enough with respect to the provided values of both transmit/receive buffer descriptors.</p> <p>If the caller provides the shared memory region, then the driver assumes that this region does not require cache coherency operations, nor does it require conversions between virtual and physical addresses.</p> <p>If the caller indicates that this routine must allocate the shared memory region, then this routine will use <i>cacheDmaMalloc()</i> to obtain some cache-safe memory. The attributes of this memory will be checked, and if the memory is not write coherent, this routine will abort and return NULL.</p> |
| RETURNS | an END object pointer, or NULL on error. |
| SEE ALSO | motFecEnd, ifLib , <i>MPC860T Fast Ethernet Controller (Supplement to MPC860 User's Manual)</i> |

mountdInit()

NAME *mountdInit()* – initialize the mount daemon

SYNOPSIS

```
STATUS mountdInit
(
    int    priority, /* priority of the mount daemon */
    int    stackSize, /* stack size of the mount daemon */
    FUNCPTR authHook, /* hook to run to authorize each request */
    int    nExports, /* maximum number of exported file systems */
    int    options /* currently unused - set to 0 */
)
```

DESCRIPTION This routine spawns a mount daemon if one does not already exist. Defaults for the *priority* and *stackSize* arguments are in the global variables **mountdPriorityDefault** and **mountdStackSizeDefault**, and are initially set to MOUNTD_PRIORITY_DEFAULT and MOUNTD_STACKSIZE_DEFAULT respectively.

Normally, no authorization checking is performed by either mountd or nfsd. To add authorization checking, set *authHook* to point to a routine declared as follows:

```
nfsstat routine
(
    int          progNum, /* RPC program number */
    int          versNum, /* RPC program version number */
    int          procNum, /* RPC procedure number */
    struct sockaddr_in clientAddr, /* address of the client */
    MOUNTD_ARGUMENT * mountdArg /* argument of the call */
)
```

The *authHook* callback must return OK if the request is authorized, and any defined NFS error code (usually NFSERR_ACCES) if not.

RETURNS OK, or ERROR if the mount daemon could not be correctly initialized.

SEE ALSO **mountLib**

mqPxLibInit()

NAME *mqPxLibInit()* – initialize the POSIX message queue library

SYNOPSIS

```
int mqPxLibInit
(
    int hashSize /* log2 of number of hash buckets */
)
```

DESCRIPTION This routine initializes the POSIX message queue facility. If *hashSize* is 0, the default value is taken from `MQ_HASH_SIZE_DEFAULT`.

RETURNS OK or ERROR.

SEE ALSO `mqPxLib`

mqPxShowInit()

NAME *mqPxShowInit()* – initialize the POSIX message queue show facility

SYNOPSIS

```
STATUS mqPxShowInit (void)
```

DESCRIPTION This routine links the POSIX message queue show routine into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
- If you use the Tornado project facility, select `INCLUDE_POSIX_MQ_SHOW`.

RETURNS OK, or ERROR if an error occurs installing the file pointer show routine.

SEE ALSO `mqPxShow`

mq_close()

mq_close()

| | |
|--------------------|---|
| NAME | <i>mq_close()</i> – close a message queue (POSIX) |
| SYNOPSIS | <pre>int mq_close (mqd_t mqdes /* message queue descriptor */)</pre> |
| DESCRIPTION | This routine is used to indicate that the calling task is finished with the specified message queue <i>mqdes</i> . The <i>mq_close()</i> call deallocates any system resources allocated by the system for use by this task for its message queue. The behavior of a task that is blocked on either a <i>mq_send()</i> or <i>mq_receive()</i> is undefined when <i>mq_close()</i> is called. The <i>mqdes</i> parameter will no longer be a valid message queue ID. |
| RETURNS | 0 (OK) if the message queue is closed successfully, otherwise -1 (ERROR). |
| ERRNO | EBADF |
| SEE ALSO | <i>mqPxBLib</i> , <i>mq_open()</i> |

mq_getattr()

| | |
|--------------------|--|
| NAME | <i>mq_getattr()</i> – get message queue attributes (POSIX) |
| SYNOPSIS | <pre>int mq_getattr (mqd_t mqdes, /* message queue descriptor */ struct mq_attr * pMqStat /* buffer in which to return attributes */)</pre> |
| DESCRIPTION | This routine gets status information and attributes associated with a specified message queue <i>mqdes</i> . Upon return, the following members of the mq_attr structure referenced by <i>pMqStat</i> will contain the values set when the message queue was created but with modifications made by subsequent calls to <i>mq_setattr()</i> : <p>mq_flags May be modified by <i>mq_setattr()</i>.</p> <p>The following were set at message queue creation:</p> <p>mq_maxmsg Maximum number of messages.</p> |

mq_msgsize

Maximum message size.

mq_curmsgs

The number of messages currently in the queue.

RETURNS 0 (OK) if message attributes can be determined, otherwise -1 (ERROR).**ERRNO** EBADF**SEE ALSO** *mqPxBLib, mq_open(), mq_send(), mq_setattr()*

mq_notify()

NAME *mq_notify()* – notify a task that a message is available on a queue (POSIX)**SYNOPSIS**

```
int mq_notify
(
    mqd_t          mqdes,          /* message queue descriptor */
    const struct sigevent * pNotification /* real-time signal */
)
```

DESCRIPTION If *pNotification* is not NULL, this routine attaches the specified *pNotification* request by the calling task to the specified message queue *mqdes* associated with the calling task. The real-time signal specified by *pNotification* will be sent to the task when the message queue changes from empty to non-empty. If a task has already attached a notification request to the message queue, all subsequent attempts to attach a notification to the message queue will fail. A task is able to attach a single notification to each *mqdes* it has unless another task has already attached one.If *pNotification* is NULL and the task has previously attached a notification request to the message queue, the attached notification request is detached and the queue is available for another task to attach a notification request.If a notification request is attached to a message queue and any task is blocked in *mq_receive()* waiting to receive a message when a message arrives at the queue, then the appropriate *mq_receive()* will be completed and the notification request remains pending.**RETURNS** 0 (OK) if successful, otherwise -1 (ERROR).**ERRNO** EBADF, EBUSY, EINVAL**SEE ALSO** *mqPxBLib, mq_open(), mq_send()*

mq_open()

NAME *mq_open()* – open a message queue (POSIX)

SYNOPSIS

```
mqd_t mq_open
(
    const char * mqName, /* name of queue to open */
    int         oflags /* open flags */
)
```

DESCRIPTION This routine establishes a connection between a named message queue and the calling task. After a call to *mq_open()*, the task can reference the message queue using the address returned by the call. The message queue remains usable until the queue is closed by a successful call to *mq_close()*.

The *oflags* argument controls whether the message queue is created or merely accessed by the *mq_open()* call. The following flag bits can be set in *oflags*:

O_RDONLY

Open the message queue for receiving messages. The task can use the returned message queue descriptor with *mq_receive()*, but not *mq_send()*.

O_WRONLY

Open the message queue for sending messages. The task can use the returned message queue descriptor with *mq_send()*, but not *mq_receive()*.

O_RDWR

Open the queue for both receiving and sending messages. The task can use any of the functions allowed for **O_RDONLY** and **O_WRONLY**.

Any combination of the remaining flags can be specified in *oflags*:

O_CREAT

This flag is used to create a message queue if it does not already exist. If **O_CREAT** is set and the message queue already exists, then **O_CREAT** has no effect except as noted below under **O_EXCL**. Otherwise, *mq_open()* creates a message queue. The **O_CREAT** flag requires a third and fourth argument: *mode*, which is of type **mode_t**, and *pAttr*, which is of type pointer to an **mq_attr** structure. The value of *mode* has no effect in this implementation. If *pAttr* is NULL, the message queue is created with implementation-defined default message queue attributes. If *pAttr* is non-NULL, the message queue attributes **mq_maxmsg** and **mq_msgsize** are set to the values of the corresponding members in the **mq_attr** structure referred to by *pAttr*; if either attribute is less than or equal to zero, an error is returned and **errno** is set to **EINVAL**.

O_EXCL

This flag is used to test whether a message queue already exists. If **O_EXCL** and **O_CREAT** are set, *mq_open()* fails if the message queue name exists.

O_NONBLOCK

The setting of this flag is associated with the open message queue descriptor and determines whether a *mq_send()* or *mq_receive()* will wait for resources or messages that are not currently available, or fail with *errno* set to *EAGAIN*.

The *mq_open()* call does not add or remove messages from the queue.

NOTE

Some POSIX functionality is not yet supported:

- A message queue cannot be closed with calls to *_exit()* or *exec()*.
- A message queue cannot be implemented as a file.
- Message queue names will not appear in the file system.

RETURNS

A message queue descriptor, otherwise -1 (ERROR).

ERRNO

EEXIST, *EINVAL*, *ENOENT*, *ENOSPC*

SEE ALSO

mqPxLib, *mq_send()*, *mq_receive()*, *mq_close()*, *mq_setattr()*, *mq_getattr()*, *mq_unlink()*

mq_receive()

NAME

mq_receive() – receive a message from a message queue (POSIX)

SYNOPSIS

```
ssize_t mq_receive
(
    mqd_t mqdes, /* message queue descriptor */
    void * pMsg, /* buffer to receive message */
    size_t msgLen, /* size of buffer, in bytes */
    int * pMsgPrio /* if not NULL, priority of message */
)
```

DESCRIPTION

This routine receives the oldest of the highest priority message from the message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msgLen* argument, is less than the *mq_msgsize* attribute of the message queue, *mq_receive()* will fail and return an error. Otherwise, the selected message is removed from the queue and copied to *pMsg*.

If *pMsgPrio* is not NULL, the priority of the selected message will be stored in *pMsgPrio*.

If the message queue is empty and *O_NONBLOCK* is not set in the message queue's description, *mq_receive()* will block until a message is added to the message queue, or until it is interrupted by a signal. If more than one task is waiting to receive a message when a message arrives at an empty queue, the task of highest priority that has been waiting the longest will be selected to receive the message. If the specified message queue

mq_send()

is empty and `O_NONBLOCK` is set in the message queue's description, no message is removed from the queue, and `mq_receive()` returns an error.

| | |
|-----------------|--|
| RETURNS | The length of the selected message in bytes, otherwise -1 (ERROR). |
| ERRNO | EAGAIN, EBADF, EMSGSIZE, EINTR |
| SEE ALSO | <code>mqPxLib</code> , <code>mq_send()</code> |

mq_send()

NAME `mq_send()` – send a message to a message queue (POSIX)

SYNOPSIS

```
int mq_send
(
    mqd_t      mqdes, /* message queue descriptor */
    const void * pMsg, /* message to send */
    size_t     msgLen, /* size of message, in bytes */
    int        msgPrio /* priority of message */
)
```

DESCRIPTION This routine adds the message `pMsg` to the message queue `mqdes`. The `msgLen` parameter specifies the length of the message in bytes pointed to by `pMsg`. The value of `pMsg` must be less than or equal to the `mq_msgsize` attribute of the message queue, or `mq_send()` will fail.

If the message queue is not full, `mq_send()` will behave as if the message is inserted into the message queue at the position indicated by the `msgPrio` argument. A message with a higher numeric value for `msgPrio` is inserted before messages with a lower value. The value of `msgPrio` must be less than or equal to 31.

If the specified message queue is full and `O_NONBLOCK` is not set in the message queue's, `mq_send()` will block until space becomes available to queue the message, or until it is interrupted by a signal. The priority scheduling option is supported in the event that there is more than one task waiting on space becoming available. If the message queue is full and `O_NONBLOCK` is set in the message queue's description, the message is not queued, and `mq_send()` returns an error.

USE BY INTERRUPT SERVICE ROUTINES

This routine can be called by interrupt service routines as well as by tasks. This is one of the primary means of communication between an interrupt service routine and a task. If `mq_send()` is called from an interrupt service routine, it will behave as if the `O_NONBLOCK` flag were set.

RETURNS 0 (OK), otherwise -1 (ERROR).

ERRNO EAGAIN, EBADF, EINTR, EINVAL, EMSGSIZE

SEE ALSO mqPxLib, mq_receive()

mq_setattr()

NAME *mq_setattr()* – set message queue attributes (POSIX)

SYNOPSIS

```
int mq_setattr
(
    mqd_t          mqdes,      /* message queue descriptor */
    const struct mq_attr * pMqStat, /* new attributes */
    struct mq_attr *   pOldMqStat /* old attributes */
)
```

DESCRIPTION This routine sets attributes associated with the specified message queue *mqdes*. The message queue attributes corresponding to the following members defined in the **mq_attr** structure are set to the specified values upon successful completion of the call:

mq_flags

The value the **O_NONBLOCK** flag.

If *pOldMqStat* is non-NULL, *mq_setattr()* will store, in the location referenced by *pOldMqStat*, the previous message queue attributes and the current queue status. These values are the same as would be returned by a call to *mq_getattr()* at that point.

RETURNS 0 (OK) if attributes are set successfully, otherwise -1 (ERROR).

ERRNO EBADF

SEE ALSO mqPxLib, mq_open(), mq_send(), mq_getattr()

mq_unlink()

mq_unlink()**NAME** *mq_unlink()* – remove a message queue (POSIX)**SYNOPSIS**

```
int mq_unlink
(
    const char * mqName /* name of message queue */
)
```

DESCRIPTION This routine removes the message queue named by the pathname *mqName*. After a successful call to *mq_unlink()*, a call to *mq_open()* on the same message queue will fail if the flag **O_CREAT** is not set. If one or more tasks have the message queue open when *mq_unlink()* is called, removal of the message queue is postponed until all references to the message queue have been closed.**RETURNS** 0 (OK) if the message queue is unlinked successfully, otherwise -1 (ERROR).**ERRNO** ENOENT**SEE ALSO** *mqPxLib*, *mq_close()*, *mq_open()*

mRegs()**NAME** *mRegs()* – modify registers**SYNOPSIS**

```
STATUS mRegs
(
    char * regName, /* register name, NULL for all */
    int taskNameOrId /* task name or task ID, 0 = default task */
)
```

DESCRIPTION This command modifies the specified register for the specified task. If *taskNameOrId* is omitted or zero, the last task referenced is assumed. If the specified register is not found, it prints out the valid register list and returns ERROR. If no register is specified, it sequentially prompts the user for new values for a task's registers. It displays each register and the current contents of that register, in turn. The user can respond in one of several ways:**RETURN**

Do not change this register, but continue, prompting at the next register.

number

Set this register to *number*.

. (dot)

Do not change this register, and quit.

EOF

Do not change this register, and quit.

All numbers are entered and displayed in hexadecimal, except floating-point values, which may be entered in double precision.

RETURNS OK, or ERROR if the task or register does not exist.

SEE ALSO **usrLib**, **m()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

mRouteAdd()

NAME *mRouteAdd()* – add multiple routes to the same destination

SYNOPSIS

```
STATUS mRouteAdd
(
    char * pDest, /* destination addr in internet dot notation */
    char * pGate, /* gateway address in internet dot notation */
    long  mask,   /* mask for destination */
    int   tos,    /* type of service */
    int   flags  /* route flags */
)
```

DESCRIPTION This routine is similar to *routeAdd()*, except that you can use multiple *mRouteAdd()* calls to add multiple routes to the same location. Use *pDest* to specify the destination, *pGate* to specify the gateway to that destination, *mask* to specify destination mask, and *tos* to specify the type of service. For *tos*, **netinet/ip.h** defines the following constants as valid values:

```
    IPTOS_LOWDELAY
    IPTOS_THROUGHPUT
    IPTOS_RELIABILITY
    IPTOS_MINCOST
```

Use *flags* to specify any flags you want to associate with this entry. The valid non-zero values are **RTF_HOST** and **RTF_CLONING** defined in **net/route.h**.

EXAMPLE To add a route to the 90.0.0.0 network through 91.0.0.3:

mRouteDelete()

```
-> mRouteAdd ("90.0.0.0", "91.0.0.3", 0xffffffff00, 0, 0);
```

Using *mRouteAdd()*, you could create multiple routes to the same destination. VxWorks would distinguish among these routes based on factors such as the netmask or the type of service. Thus, it is perfectly legal to say:

```
-> mRouteAdd ("90.0.0.0", "91.0.0.3", 0xffffffff00, 0, 0);
-> mRouteAdd ("90.0.0.0", "91.0.0.254", 0xffff0000, 0, 0);
```

This adds two routes to the same network, "90.0.0.0", that go by two different gateways. The differentiating factor is the netmask.

This routine adds a route of type **M2_ipRouteProto_other**, which is a static route. This route will not be modified or deleted until a call to *mRouteDelete()* removes it.

RETURNS

OK or ERROR.

SEE ALSO

routeLib, *mRouteEntryAdd()*, *mRouteDelete()*, *routeAdd()*

mRouteDelete()

NAME

mRouteDelete() – delete a route from the routing table

SYNOPSIS

```
STATUS mRouteDelete
(
    char * pDest, /* destination address */
    long  mask,  /* mask for destination */
    int   tos,   /* type of service */
    int   flags /* either 0 or RTF_HOST */
)
```

DESCRIPTION

This routine deletes a routing table entry as specified by the destination, *pDest*, the destination mask, *mask*, and type of service, *tos*. The *tos* values are as defined in the reference entry for *mRouteAdd()*.

EXAMPLE

Consider the case of a route added in the following manner:

```
-> mRouteAdd ("90.0.0.0", "91.0.0.3", 0xffffffff00, 0, 0);
```

To delete a route that was added in the above manner, call *mRouteDelete()* as follows:

```
-> mRouteDelete("90.0.0.0", 0xffffffff00, 0);
```

If the netmask and or type of service do not match, the route is not deleted.

The value of *flags* should be *RTF_HOST* for host routes, *RTF_CLONING* for routes which need to be cloned, and 0 in all other cases.

RETURNS OK or ERROR.

SEE ALSO routeLib, mRouteAdd()

mRouteEntryAdd()

NAME *mRouteEntryAdd()* – add a protocol-specific route to the routing table

SYNOPSIS

```
STATUS mRouteEntryAdd
(
    long destIp, /* destination address, network order */
    long gateIp, /* gateway address, network order */
    long mask,   /* mask for destination, network order */
    int  tos,    /* type of service */
    int  flags,  /* route flags */
    int  proto   /* routing protocol */
)
```

DESCRIPTION For a single destination *destIp*, this routine can add additional routes *gateIp* to the routing table. The different routes are distinguished by a destination mask *mask*, the type of service *tos*, and associated flag values *flags*. Valid values for *flags* are 0, RTF_HOST, RTF_CLONING (defined in **net/route.h**). The *proto* parameter identifies the protocol that generated this route. Values for *proto* may be found in **m2Lib.h**. The *tos* parameter takes one of following values (defined in **netinet/ip.h**):

```
    IPTOS_LOWDELAY
    IPTOS_THROUGHPUT
    IPTOS_RELIABILITY
    IPTOS_MINCOST
```

RETURNS OK or ERROR.

SEE ALSO routeLib, m2Lib.h, mRouteAdd(), mRouteDelete()

mRouteEntryDelete()

NAME *mRouteEntryDelete()* – delete route from the routing table

SYNOPSIS

```
STATUS mRouteEntryDelete
(
    long destIp, /* destination address, network order */
    long gateIp, /* gateway address, network order */
    long mask,   /* mask for destination, network order */
    int  tos,    /* type of service */
    int  flags,  /* route flags */
    int  proto   /* routing protocol */
)
```

DESCRIPTION This routine deletes a protocol-specific route from the routing table. Specify the route using a destination *pDest*, a gateway *pGate*, a destination mask *mask*, the type of service *tos*, a *flags* value, and a *proto* value that identifies the routing protocol that added the route. The valid values for *flags* are 0 and **RTF_HOST** (defined in **net/route.h**). Values for *proto* may be found in **m2Lib.h** and *tos* is one of the following values defined in **netinet/ip.h**:

```
    IPTOS_LOWDELA
    IPTOS_THROUGHPU
    IPTOS_RELIABILIT
    IPTOS_MINCOST
```

An existing route is deleted only if it is owned by the protocol specified by *proto*.

RETURNS OK or ERROR.

SEE ALSO **routeLib**

mRouteShow()

NAME *mRouteShow()* – print the entries of the routing table

SYNOPSIS

```
void mRouteShow
(
)
```

DESCRIPTION This routine prints the route entries in the routing table.

RETURNS N/A

SEE ALSO netShow

msgQCreate()

NAME *msgQCreate()* – create and initialize a message queue

SYNOPSIS

```
MSG_Q_ID msgQCreate
(
    int maxMsgs,          /* max messages that can be queued */
    int maxMsgLength,    /* max bytes in a message */
    int options           /* message queue options */
)
```

DESCRIPTION This routine creates a message queue capable of holding up to *maxMsgs* messages, each up to *maxMsgLength* bytes long. The routine returns a message queue ID used to identify the created message queue in all subsequent calls to routines in this library. The queue can be created with the following options:

MSG_Q_FIFO (0x00)
queue pending tasks in FIFO order.

MSG_Q_PRIORITY (0x01)
queue pending tasks in priority order.

RETURNS *MSG_Q_ID*, or NULL if error.

ERRNO *S_memLib_NOT_ENOUGH_MEMORY*, *S_intLib_NOT_ISR_CALLABLE*

SEE ALSO *msgQLib*, *msgQSmLib*

msgQDelete()

NAME *msgQDelete()* – delete a message queue

SYNOPSIS

```
STATUS msgQDelete
(
    MSG_Q_ID msgQId /* message queue to delete */
)
```

msgQInfoGet()

| | |
|--------------------|---|
| DESCRIPTION | This routine deletes a message queue. Any task blocked on either a <i>msgQSend()</i> or <i>msgQReceive()</i> will be unblocked and receive an error from the call with errno set to S_objLib_OBJECT_DELETED . The <i>msgQId</i> parameter will no longer be a valid message queue ID. |
| RETURNS | OK or ERROR. |
| ERRNO | S_objLib_OBJ_ID_ERROR, S_intLib_NOT_ISR_CALLABLE |
| SEE ALSO | msgQLib, msgQSmLib |

msgQInfoGet()

NAME *msgQInfoGet()* – get information about a message queue

SYNOPSIS

```
STATUS msgQInfoGet
(
    MSG_Q_ID    msgQId, /* message queue to query */
    MSG_Q_INFO * pInfo  /* where to return msg info */
)
```

DESCRIPTION This routine gets information about the state and contents of a message queue. The parameter *pInfo* is a pointer to a structure of type **MSG_Q_INFO** defined in **msgQLib.h** as follows:

```
typedef struct          /* MSG_Q_INFO */
{
    int    numMsgs;      /* OUT: number of messages queued      */
    int    numTasks;    /* OUT: number of tasks waiting on msg q */
    int    sendTimeouts; /* OUT: count of send timeouts        */
    int    rcvTimeouts; /* OUT: count of receive timeouts     */
    int    options;     /* OUT: options with which msg q was created */
    int    maxMsgs;     /* OUT: max messages that can be queued */
    int    maxMsgLength; /* OUT: max byte length of each message */
    int    taskIdListMax; /* IN: max tasks to fill in taskIdList */
    int *  taskIdList;  /* PTR: array of task IDs waiting on msg q */
    int    msgListMax;  /* IN: max msgs to fill in msg lists    */
    char ** msgPtrList; /* PTR: array of msg ptrs queued to msg q */
    int *  msgLenList;  /* PTR: array of lengths of msgs      */
} MSG_Q_INFO;
```

If a message queue is empty, there may be tasks blocked on receiving. If a message queue is full, there may be tasks blocked on sending. This can be determined as follows:

- If *numMsgs* is 0, then *numTasks* indicates the number of tasks blocked on receiving.
- If *numMsgs* is equal to *maxMsgs*, then *numTasks* is the number of tasks blocked on sending.
- If *numMsgs* is greater than 0 but less than *maxMsgs*, then *numTasks* will be 0.

A list of pointers to the messages queued and their lengths can be obtained by setting *msgPtrList* and *msgLenList* to the addresses of arrays to receive the respective lists, and setting *msgListMax* to the maximum number of elements in those arrays. If either list pointer is NULL, no data will be returned for that array.

No more than *msgListMax* message pointers and lengths are returned, although *numMsgs* will always be returned with the actual number of messages queued.

For example, if the caller supplies a *msgPtrList* and *msgLenList* with room for 10 messages and sets *msgListMax* to 10, but there are 20 messages queued, then the pointers and lengths of the first 10 messages in the queue are returned in *msgPtrList* and *msgLenList*, but *numMsgs* will be returned with the value 20.

A list of the task IDs of tasks blocked on the message queue can be obtained by setting *taskIdList* to the address of an array to receive the list, and setting *taskIdListMax* to the maximum number of elements in that array. If *taskIdList* is NULL, then no task IDs are returned. No more than *taskIdListMax* task IDs are returned, although *numTasks* will always be returned with the actual number of tasks blocked.

For example, if the caller supplies a *taskIdList* with room for 10 task IDs and sets *taskIdListMax* to 10, but there are 20 tasks blocked on the message queue, then the IDs of the first 10 tasks in the blocked queue will be returned in *taskIdList*, but *numTasks* will be returned with the value 20.

Note that the tasks returned in *taskIdList* may be blocked for either send or receive. As noted above this can be determined by examining *numMsgs*.

The variables *sendTimeouts* and *recvTimeouts* are the counts of the number of times *msgQSend()* and *msgQReceive()* respectively returned with a timeout.

The variables *options*, *maxMsgs*, and *maxMsgLength* are the parameters with which the message queue was created.

WARNING

The information returned by this routine is not static and may be obsolete by the time it is examined. In particular, the lists of task IDs and/or message pointers may no longer be valid. However, the information is obtained atomically, thus it will be an accurate snapshot of the state of the message queue at the time of the call. This information is generally used for debugging purposes only.

WARNING

The current implementation of this routine locks out interrupts while obtaining the information. This can compromise the overall interrupt latency of the system. Generally this routine is used for debugging purposes only.

RETURNS OK or ERROR.

ERRNO S_distLib_NOT_INITIALIZED, S_smObjLib_NOT_INITIALIZED, S_objLib_OBJ_ID_ERROR

SEE ALSO msgQShow

msgQNumMsgs()

NAME *msgQNumMsgs()* – get the number of messages queued to a message queue

SYNOPSIS

```
int msgQNumMsgs
(
    MSG_Q_ID msgQId /* message queue to examine */
)
```

DESCRIPTION This routine returns the number of messages currently queued to a specified message queue.

RETURNS The number of messages queued, or ERROR.

ERRNO S_distLib_NOT_INITIALIZED, S_smObjLib_NOT_INITIALIZED, S_objLib_OBJ_ID_ERROR

SEE ALSO msgQLib, msgQSmLib

msgQReceive()

NAME *msgQReceive()* – receive a message from a message queue

SYNOPSIS

```
int msgQReceive
(
    MSG_Q_ID msgQId, /* message queue from which to receive */
    char * buffer, /* buffer to receive message */
    UINT maxNBytes, /* length of buffer */
    int timeout /* ticks to wait */
)
```

DESCRIPTION This routine receives a message from the message queue *msgQId*. The received message is copied into the specified *buffer*, which is *maxNBytes* in length. If the message is longer

than *maxNBytes*, the remainder of the message is discarded (no error indication is returned).

The *timeout* parameter specifies the number of ticks to wait for a message to be sent to the queue, if no message is available when *msgQReceive()* is called. The *timeout* parameter can also have the following special values:

NO_WAIT (0)

return immediately, even if the message has not been sent.

WAIT_FOREVER (-1)

never time out.

WARNING This routine must not be called by interrupt service routines.

RETURNS The number of bytes copied to *buffer*, or ERROR.

ERRNO *S_distLib_NOT_INITIALIZED*, *S_smObjLib_NOT_INITIALIZED*, *S_objLib_OBJ_ID_ERROR*, *S_objLib_OBJ_DELETED*, *S_objLib_OBJ_UNAVAILABLE*, *S_objLib_OBJ_TIMEOUT*, *S_msgQLib_INVALID_MSG_LENGTH*

SEE ALSO *msgQLib*, *msgQSmLib*

msgQSend()

NAME *msgQSend()* – send a message to a message queue

SYNOPSIS `STATUS msgQSend`

```
(
    MSG_Q_ID msgQId, /* message queue on which to send */
    char *   buffer, /* message to send */
    UINT     nBytes, /* length of message */
    int      timeout, /* ticks to wait */
    int      priority /* MSG_PRI_NORMAL or MSG_PRI_URGENT */
)
```

DESCRIPTION This routine sends the message in *buffer* of length *nBytes* to the message queue *msgQId*. If any tasks are already waiting to receive messages on the queue, the message will immediately be delivered to the first waiting task. If no task is waiting to receive messages, the message is saved in the message queue.

The *timeout* parameter specifies the number of ticks to wait for free space if the message queue is full. The *timeout* parameter can also have the following special values:

NO_WAIT (0)

return immediately, even if the message has not been sent.

WAIT_FOREVER (-1)

never time out.

The *priority* parameter specifies the priority of the message being sent. The possible values are:

MSG_PRI_NORMAL (0)

normal priority; add the message to the tail of the list of queued messages.

MSG_PRI_URGENT (1)

urgent priority; add the message to the head of the list of queued messages.

USE BY INTERRUPT SERVICE ROUTINES

This routine can be called by interrupt service routines as well as by tasks. This is one of the primary means of communication between an interrupt service routine and a task. When called from an interrupt service routine, *timeout* must be **NO_WAIT**.

RETURNS

OK or ERROR.

ERRNO

S_distLib_NOT_INITIALIZED, **S_objLib_OBJ_ID_ERROR**, **S_objLib_OBJ_DELETED**,
S_objLib_OBJ_UNAVAILABLE, **S_objLib_OBJ_TIMEOUT**,
S_msgQLib_INVALID_MSG_LENGTH, **S_msgQLib_NON_ZERO_TIMEOUT_AT_INT_LEVEL**

SEE ALSO

msgQLib, **msgQSmLib**

msgQShow()

NAME

msgQShow() – show information about a message queue

SYNOPSIS

```
STATUS msgQShow  
(  
    MSG_Q_ID msgQId, /* message queue to display */  
    int      level   /* 0 = summary, 1 = details */  
)
```

DESCRIPTION

This routine displays the state and optionally the contents of a message queue.

A summary of the state of the message queue is displayed as follows:

```
Message Queue Id      : 0x3f8c20  
Task Queuing          : FIFO  
Message Byte Len     : 150
```

```

Messages Max      : 50
Messages Queued   : 0
Receivers Blocked : 1
Send timeouts     : 0
Receive timeouts  : 0

```

If *level* is 1, then more detailed information will be displayed. If messages are queued, they will be displayed as follows:

```

Messages queued:
#   address length value
1 0x123eb204   4 0x00000001 0x12345678

```

If tasks are blocked on the queue, they will be displayed as follows:

```

Receivers blocked:
NAME      TID     PRI  DELAY
-----
tExcTask  3fd678   0    21

```

- RETURNS** OK or ERROR.
- ERRNO** S_distLib_NOT_INITIALIZED, S_smObjLib_NOT_INITIALIZED
- SEE ALSO** **msgQShow**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

msgQShowInit()

- NAME** *msgQShowInit()* – initialize the message queue show facility
- SYNOPSIS** **void msgQShowInit (void)**
- DESCRIPTION** This routine links the message queue show facility into the VxWorks system. It is called automatically when the message queue show facility is configured into VxWorks using either of the following methods:
- If you use the configuration header files, define **INCLUDE_SHOW_ROUTINES** in **config.h**.
 - If you use the Tornado project facility, select **INCLUDE_MSG_Q_SHOW**.
- RETURNS** N/A
- SEE ALSO** **msgQShow**

msgQSmCreate()

NAME *msgQSmCreate()* – create and initialize a shared memory message queue (VxMP Opt.)

SYNOPSIS

```
MSG_Q_ID msgQSmCreate
(
    int maxMsgs,          /* max messages that can be queued */
    int maxMsgLength,    /* max bytes in a message */
    int options          /* message queue options */
)
```

DESCRIPTION This routine creates a shared memory message queue capable of holding up to *maxMsgs* messages, each up to *maxMsgLength* bytes long. It returns a message queue ID used to identify the created message queue. The queue can only be created with the option `MSG_Q_FIFO` (0), thus queuing pending tasks in FIFO order.

The global message queue identifier returned can be used directly by generic message queue handling routines in `msgQLib` -- *msgQSend()*, *msgQReceive()*, and *msgQNumMsgs()* -- and by the show routines *show()* and *msgQShow()*.

If there is insufficient memory to store the message queue structure in the shared memory message queue partition or if the shared memory system pool cannot handle the requested message queue size, shared memory message queue creation will fail with **errno** set to `S_memLib_NOT_ENOUGH_MEMORY`. This problem can be solved by incrementing the value of `SM_OBJ_MAX_MSG_Qand`/or the shared memory objects dedicated memory size `SM_OBJ_MEM_SIZE`.

Before this routine can be called, the shared memory objects facility must be initialized (see `msgQSmLib`).

AVAILABILITY This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

RETURNS `MSG_Q_ID`, or NULL if error.

ERRNO `S_memLib_NOT_ENOUGH_MEMORY`, `S_intLib_NOT_ISR_CALLABLE`,
`S_msgQLib_INVALID_QUEUE_TYPE`, `S_smObjLib_LOCK_TIMEOUT`

SEE ALSO `msgQSmLib`, `smObjLib`, `msgQLib`, `msgQShow`

munlock()

NAME *munlock()* – unlock specified pages (POSIX)

SYNOPSIS

```
int munlock
(
    const void * addr,
    size_t      len
)
```

DESCRIPTION This routine unlocks specified pages from being memory resident.

RETURNS 0 (OK) always.

ERRNO N/A

SEE ALSO *mmanPxLib*

munlockall()

NAME *munlockall()* – unlock all pages used by a process (POSIX)

SYNOPSIS

```
int munlockall (void)
```

DESCRIPTION This routine unlocks all pages used by a process from being memory resident.

RETURNS 0 (OK) always.

ERRNO N/A

SEE ALSO *mmanPxLib*

muxAddressForm()

NAME *muxAddressForm()* – form an address into a packet

SYNOPSIS

```
M_BLK_ID muxAddressForm
(
    void*   pCookie, /* cookie that identifies the device */
    M_BLK_ID pMblk,  /* structure to contain packet */
    M_BLK_ID pSrcAddr, /* structure containing source address */
    M_BLK_ID pDstAddr /* structure containing destination address */
)
```

DESCRIPTION This routine accepts the source and destination addressing information through the *pSrcAddr* and *pDstAddr* mBlks and returns an **M_BLK_ID** that points to the assembled link-level header. This routine prepends the link-level header into *pMblk* if there is enough space available or it allocates a new **mBlk-clBlk**-cluster and prepends the new *mBlk* to the **mBlk** chain passed in *pMblk*. This routine returns a pointer to an **mBlk** that contains the link-level header information.

pCookie

Expects the pointer returned from the *muxBind()*. This pointer identifies the device to which the MUX has bound this protocol.

pMblk

Expects a pointer to the **mBlk** structure that contains the packet.

pSrcAddr

Expects a pointer to the **mBlk** that contains the source address.

pDstAddr

Expects a pointer to the **mBlk** that contains the destination address.

RETURNS **M_BLK_ID** or **NULL**.

ERRNO **S_muxLib_NO_DEVICE**

SEE ALSO **muxLib**

muxAddrResFuncAdd()

NAME *muxAddrResFuncAdd()* – add an address resolution function

SYNOPSIS

```
STATUS muxAddrResFuncAdd
(
    long    ifType,    /* Media interface type from m2Lib.h */
    long    protocol, /* Protocol type from RFC 1700 */
    FUNCPTR addrResFunc /* Function to call. */
)
```

DESCRIPTION This routine takes an *ifType* from **m2Lib.h**, a protocol number from RFC 1700 and a pointer to an address resolution function and installs that function for later retrieval by *muxAddrResFuncGet()*.

ifType

Expects a media interface or network driver type from **m2Lib.h**

protocol

Expects a network service or protocol type from RFC 1700

addrResFunc

Expects a pointer to an address resolution function for this driver and protocol

RETURNS OK or ERROR.

SEE ALSO **muxLib**

muxAddrResFuncDel()

NAME *muxAddrResFuncDel()* – delete an address resolution function

SYNOPSIS

```
STATUS muxAddrResFuncDel
(
    long ifType, /* ifType of function you want to delete */
    long protocol /* protocol from which to delete the function */
)
```

DESCRIPTION This function takes an *ifType* (from **m2Lib.h**) and a protocol (from RFC 1700) and deletes the associated address resolution routine (if such exists).

ifType

Expects a media interface or network driver type from **m2Lib.h**

protocol

Expects a network service or protocol type from RFC 1700

RETURNS OK or ERROR.

SEE ALSO **muxLib**

muxAddrResFuncGet()

NAME ***muxAddrResFuncGet()*** – get the address resolution function for *ifType*/protocol

SYNOPSIS

```
FUNCPTR muxAddrResFuncGet  
(  
    long ifType, /* ifType from m2Lib.h */  
    long protocol /* protocol from RFC 1700 */  
)
```

DESCRIPTION This routine takes an *ifType* (from **m2Lib.h**) and a protocol (from RFC 1700) and returns a pointer to the address resolution function registered for this *ifType*/protocol pair. If no such function exists then NULL is returned.

ifType

Expects a media interface or network driver type from **m2Lib.h**

protocol

Expects a network service or protocol type from RFC 1700

RETURNS FUNCPTR to the routine or NULL.

SEE ALSO **muxLib**

muxBind()

NAME *muxBind()* – bind a protocol to the MUX given a driver name

SYNOPSIS

```
END_OBJ* muxBind
(
  char *                pName, /* interface name, for example, ln, */
  int                   unit,  /* unit number */
  BOOL (* stackRcvRtn) (void*
)
```

DESCRIPTION A protocol uses this routine to bind to a specific driver. The driver is specified by the *pName* and *unit* arguments (for example, ln and 0, ln and 1, ei and 0, ...). The *stackRcvRtn* is called whenever the MUX has a packet of the specified type. If the type is **MUX_PROTO_PROMISC**, the protocol is considered promiscuous and will get all of the packets that the MUX sees.

pName

Expects a pointer to a character string that contains the name of the device to which this protocol wants to use to send and receive packets.

unit

Expects a number which is the unit of the device of the type indicated by *pName*.

stackRcvRtn

Expects a pointer function that the MUX can call when it wants to pass a packet up to the protocol. For a description of how you should write this routine, see the description of a *stackRcvRtn()* provided in *Network Protocol Toolkit User's Guide*.

stackShutdownRtn

Expects a pointer to the function that the MUX can call to shutdown the protocol. For a description of how to write such a routine, see *stackShutdownRtn()* see the description of a *stackRcvRtn()* provided in *Network Protocol Toolkit User's Guide*.

stackErrorRtn

Expects a pointer to the function that the MUX can call to give errors to the protocol.

type

Expects a value that indicates the protocol type. The MUX uses this type to prioritize the protocol. For example, a protocol of type **MUX_PROTO_SNARF** has the highest priority (see the description of protocol prioritizing provided in *Network Protocol Toolkit User's Guide: Writing an NPT Protocol*. Aside from **MUX_PROTO_SNARF** and **MUX_PROTO_PROMISC**, valid protocol types include any of the values specified in RFC1700. If the type is **MUX_PROTO_OUTPUT**, this protocol is an output protocol and all packets that are going to be output on this device are passed to the *stackRcvRtn()* routine before actually being sent down to the device. This would be useful, for instance, for a network service that needs to send packets directly to

muxDevExists()

another network service, or for loop-back testing. If the *stackRcvRtn()* returns OK, the packet is considered to have been consumed and is no longer available. An output protocol may return ERROR from its *stackRcvRtn()* in order to look at the packet without consuming it.

pProtoName

Expects a pointer to a character string for the name of this protocol. This string can be NULL, in which case a protocol name is assigned internally.

pSpare

Expects a pointer to a structure defined by the protocol. This argument is passed up to the protocol with each received packet.

| | |
|-----------------|--|
| RETURNS | A cookie identifying the network driver to which the mux has bound the protocol. |
| ERRNO | S_muxLib_NO_DEVICE, S_muxLib_ALREADY_BOUND, S_muxLib_ALLOC_FAILED |
| SEE ALSO | muxLib |

muxDevExists()

| | |
|--------------------|---|
| NAME | <i>muxDevExists()</i> – tests whether a device is already loaded into the MUX |
| SYNOPSIS | <pre> BOOL muxDevExists (char* pName, /* string containing a device name (ln, ei, ...) */ int unit /* unit number */) </pre> |
| DESCRIPTION | <p>This routine takes a string device name (for example, ln or ei) and a unit number. If this device is already known to the MUX, it returns TRUE. Otherwise, this routine returns FALSE.</p> <p><i>pName</i> Expects a pointer to a string containing the device name</p> <p><i>unit</i> Expects the unit number of the device</p> |
| RETURNS | TRUE if the device exists, else FALSE. |
| SEE ALSO | muxLib |

muxDevLoad()

| | |
|--------------------|--|
| NAME | <i>muxDevLoad()</i> – load a driver into the MUX |
| SYNOPSIS | <pre>END_OBJ* muxDevLoad (int unit, /* unit number of device */ END_OBJ* (* endLoad) (char* , void*), /* load function of the driver */ char* pInitString, /* init string for the driver */ BOOL loaning, /* we loan buffers */ void* pBSP /* for BSP group */)</pre> |
| DESCRIPTION | <p>The <i>muxDevLoad()</i> routine loads a network driver into the MUX. Internally, this routine calls the specified <i>endLoad()</i> to initialize the software state of the device. After the device is initialized, <i>muxDevStart()</i> must be called to start the device.</p> <p><i>unit</i> Expects the unit number of the device.</p> <p><i>endLoad</i> Expects a pointer to the network driver's <i>endLoad()</i> entry point.</p> <p><i>pInitString</i> Expects a pointer to an initialization string, a colon-delimited list of options. The <i>muxDevLoad()</i> routine passes this along blindly to the <i>endLoad()</i> function.</p> <p><i>loaning</i> Expects a boolean value that tells the MUX whether the driver supports buffer loaning on this device. If the low-level device cannot support buffer loaning, passing in TRUE has no effect.</p> <p><i>pBSP</i> This argument is passed blindly to the driver, which may or may not use it. It is provided so that the BSP can pass in tables of functions that the driver can use but which are specific to the particular BSP on which it runs.</p> |
| RETURNS | A pointer to the new device or NULL if an error occurred. |
| ERRNO | S_muxLib_LOAD_FAILED |
| SEE ALSO | muxLib |

muxDevStart()

| | |
|--------------------|--|
| NAME | <i>muxDevStart()</i> – start a device by calling its start routine |
| SYNOPSIS | <pre>STATUS muxDevStart (void* pCookie /* a pointer to cookie returned by muxDevLoad() */)</pre> |
| DESCRIPTION | <p>This routine starts a device that is already initialized and loaded into the MUX. Internally, <i>muxDevStart()</i> calls the device's <i>endStart()</i>, which handles registering the driver's interrupt service routine and whatever else is needed to allow the device to handle receiving and transmitting. This call to <i>endStart()</i> provides a device-dependent way to put the device into a running state.</p> <p><i>pCookie</i> Expects a pointer to the END_OBJ returned from the <i>muxDevLoad()</i> that loaded this driver into the MUX. This "cookie" is an identifier for the device.</p> |
| RETURNS | OK, ENETDOWN if <i>pCookie</i> does not represent a valid device, or ERROR if the start routine for the device fails. |
| ERRNO | S_muxLib_NO_DEVICE |
| SEE ALSO | <i>muxLib</i> |

muxDevStop()

| | |
|--------------------|---|
| NAME | <i>muxDevStop()</i> – stop a device by calling its stop routine |
| SYNOPSIS | <pre>STATUS muxDevStop (void* pCookie /* pointer to cookie that identifies the device */)</pre> |
| DESCRIPTION | <p>This routine stops the device specified in the <i>pCookie</i> parameter. Internally, <i>muxDevStop()</i> calls the device's own stop routine, thus putting the device into a stopped state in a device-dependent manner.</p> <p><i>pCookie</i> Expects the pointer returned as the function value of the <i>muxDevLoad()</i> call for this</p> |

device. This pointer identifies the device to which the MUX has bound this protocol.

| | |
|-----------------|--|
| RETURNS | OK, ENETDOWN if <i>pCookie</i> does not represent a valid device, or ERROR if the stop routine for the device fails. |
| ERRNO | S_muxLib_NO_DEVICE |
| SEE ALSO | muxLib |

muxDevUnload()

NAME *muxDevUnload()* – remove a driver from the MUX

SYNOPSIS

```
STATUS muxDevUnload
(
    char * pName, /* a string containing the name of the device */
                /* for example, ln or ei */
    int    unit  /* the unit number */
)
```

DESCRIPTION This routine unloads a driver from the MUX. This breaks any network connections an application might have open. The *stackShutdownRtn()* of each protocol bound to the END via *muxBind()* will be called. Each *stackShutdownRtn()* is expected to call *muxUnbind()* to detach from the END.

pName

Expects a pointer to a string containing the name of the device, for example **ln** or **ei**

unit

Expects the unit number of the device indicated by *pName*

RETURNS OK on success, EINVAL or ERROR if the device's registered *endUnload()* function failed, if the specified device was not found, or some other error occurred

ERRNO S_muxLib_UNLOAD_FAILED, S_muxLib_NO_DEVICE

SEE ALSO **muxLib**

muxIoctl()

NAME *muxIoctl()* – send control information to the MUX or to a device

SYNOPSIS

```
STATUS muxIoctl
(
    void*  pCookie, /* cookie identifying the device to access */
    int    cmd,     /* command to pass to ioctl */
    caddr_t data    /* data need for command in cmd */
)
```

DESCRIPTION This routine gives the protocol access to the network driver's control functions. The MUX itself can implement some of the standard control functions, so not all commands necessarily pass down to the device. Otherwise, both command and data pass down to the device unmolested.

This routine also lets the protocol change the routine that the MUX uses to pass data up to the protocol as well as the routine that the MUX uses to shutdown the protocol.

pCookie

Expects the pointer returned as the function value of *muxBind()*. The pointer identifies the device to which this protocol is bound.

cmd

Expects a value indicating the control command you want to execute. For valid *cmd* values, see the description of the *endIoctl()* routine provided in *Network Protocol Toolkit User's Guide*.

data

Expects the data or a pointer to the data needed to carry out the command specified in *cmd*.

RETURNS OK, ENETDOWN if *pCookie* does not represent a bound device, or ERROR if the command fails.

ERRNO S_muxLib_NO_DEVICE

SEE ALSO *muxLib*

muxLibInit()

| | |
|--------------------|---|
| NAME | <i>muxLibInit()</i> – initialize global state for the MUX |
| SYNOPSIS | <code>STATUS muxLibInit (void)</code> |
| DESCRIPTION | This routine initializes all global state for the MUX. |
| RETURNS | OK or ERROR. |
| SEE ALSO | <code>muxLib</code> |

muxMCastAddrAdd()

| | |
|--------------------|--|
| NAME | <i>muxMCastAddrAdd()</i> – add a multicast address to multicast table for a device |
| SYNOPSIS | <pre>STATUS muxMCastAddrAdd (void* pCookie, /* returned by the muxBind() call */ char * pAddress /* address to add to the table */)</pre> |
| DESCRIPTION | <p>This routine adds an address to the multicast table maintained for a device. Internally, this function uses <i>pCookie</i> to find the device-specific routine that handles adding an address to the device's multicast table.</p> <p><i>pCookie</i> Expects the pointer returned as the function value of the <i>muxBind()</i> call. This pointer identifies the device to which the MUX has bound this protocol.</p> <p><i>pAddress</i> Expects a pointer to a character string containing the address you want to add.</p> |
| RETURNS | OK, ENETDOWN if <i>pCookie</i> doesn't represent a valid device, or ERROR if the device's <i>endMCastAddrAdd()</i> function fails. |
| ERRNO | ENOTSUP, S_muxLib_NO_DEVICE |
| SEE ALSO | <code>muxLib</code> |

muxMCastAddrDel()

- NAME** *muxMCastAddrDel()* – delete a multicast address from a device’s multicast table
- SYNOPSIS**
- ```
STATUS muxMCastAddrDel
(
 void* pCookie, /* Returned by the muxBind() call */
 char* pAddress /* Address to delete from the table. */
)
```
- DESCRIPTION**
- This routine deletes an address from the multicast table maintained for a device. Internally, this function uses *pCookie* to find the device-specific routine that handles deleting an address from the device’s multicast table.
- pCookie*  
Expects the pointer returned as the function value of the *muxBind()* call. This pointer identifies the device to which the MUX bound this protocol.
- pAddress*  
Expects a pointer to a character string containing the address you want to delete.
- RETURNS** OK, ENETDOWN if *pCookie* does not represent a valid driver, or ERROR if the driver’s registered *endMCastAddrDel()* function fails.
- ERRNO** ENOTSUP, EINVAL, S\_muxLib\_NO\_DEVICE
- SEE ALSO** *muxLib*

---

## ***muxMCastAddrGet()***

- NAME** *muxMCastAddrGet()* – get the multicast address table from the MUX/Driver
- SYNOPSIS**
- ```
int muxMCastAddrGet
(
    void* pCookie, /* returned by the muxBind() call */
    MULTI_TABLE * pTable /* ptr to a table to be filled and returned. */
)
```
- DESCRIPTION**
- This routine expects a buffer into which it can write the list of multicast addresses for the specified device. Internally, this routine uses *pCookie* to access the device-specific routine needed to retrieve the multicast address table.

pCookie

Expects the pointer returned as the function value of the *muxBind()* call. This pointer identifies the device to which the MUX has bound this protocol.

pTable

Expects the pointer to a **MULTI_TABLE** structure. You must have allocated this structure at some time before the call to *muxMCastAddrGet()*. The **MULTI_TABLE** structure is defined in **end.h** as:

```
typedef struct multi_table
{
    int     tableLen; /* length of table in bytes */
    char    *pTable; /* pointer to entries */
} MULTI_TABLE;
```

RETURNS OK, ENETDOWN if *pCookie* does not represent a valid driver, or ERROR if the driver's registered *endMCastAddrGet()* function fails.

ERRNO S_muxLib_NO_DEVICE

SEE ALSO muxLib

muxPacketAddrGet()

NAME *muxPacketAddrGet()* – get addressing information from a packet

SYNOPSIS STATUS *muxPacketAddrGet*

```
(
    void*    pCookie, /* cookie that identifies the device */
    M_BLK_ID pMblk, /* structure to contain packet */
    M_BLK_ID pSrcAddr, /* structure containing source address */
    M_BLK_ID pDstAddr, /* structure containing destination address */
    M_BLK_ID pESrcAddr, /* structure containing the end source */
    M_BLK_ID pEDstAddr /* structure containing the end destination */
)
```

DESCRIPTION This routine takes a pointer to cookie that was handed back by *muxBind()*, an **M_BLK_ID** that came from a device and up to four **M_BLK_ID**'s that can receive data pointers.

The routine returns appropriate information on the immediate source, immediate destination, ultimate source and, ultimate destination addresses from the packet pointed to in the first **M_BLK_ID**. This routine is a pass through to the device's own routine which knows how to interpret packets that it has received.

pCookie

Expects the cookie returned from the *muxBind()* call. This cookie identifies the device to which the MUX bound this protocol.

pMblk

Expects an **M_BLK_ID** representing packet data from which the addressing information is to be extracted

pSrcAddr

Expects NULL or an **M_BLK_ID** which will hold the local source address extracted from the packet

pDstAddr

Expects NULL or an **M_BLK_ID** which will hold the local destination address extracted from the packet

pESrcAddr

Expects NULL or an **M_BLK_ID** which will hold the end source address extracted from the packet

pEDstAddr

Expects NULL or an **M_BLK_ID** which will hold the end destination address extracted from the packet

RETURNS OK or ERROR.

ERRNO **S_muxLib_NO_DEVICE**

SEE ALSO **muxLib**

muxPacketDataGet()

NAME *muxPacketDataGet()* – return the data from a packet

SYNOPSIS

```
STATUS muxPacketDataGet
(
    void*          pCookie,    /* cookie that identifies the device */
    M_BLK_ID      pMblk,      /* returns the packet data */
    LL_HDR_INFO * pLinkHdrInfo /* the new data is returned here */
)
```

DESCRIPTION This routine copies the header information from the packet referenced in *pMblk* into the **LL_HDR_INFO** structure referenced in *pLinkHdrInfo*.

pCookie

Expects the cookie returned from the *muxBind()* call. This cookie identifies the device to which the MUX bound this protocol.

pMblk

Expects a pointer to an **mBlk** or **mBlk** cluster representing a packet containing the data to be returned

pLinkHdrInfo

Expects a pointer to an **LL_HDR_INFO** structure into which the packet header information is copied from the incoming **mBlk**

RETURNS OK or ERROR if the device type is not recognized.

ERRNO S_muxLib_NO_DEVICE

SEE ALSO muxLib

muxPollReceive()

NAME *muxPollReceive()* – poll for a packet from a device driver

SYNOPSIS

```
STATUS muxPollReceive
(
    void*    pCookie, /* cookie passed in endLoad call */
    M_BLK_ID pNBuf    /* a vector of buffers passed to us */
)
```

DESCRIPTION This is the routine that an upper layer can call to poll for a packet.

pCookie

Expects the cookie that was returned from *muxBind()*. This "cookie" is an identifier for the driver.

pNBuf

Expects a pointer to a buffer chain into which incoming data will be put.

RETURNS OK, ENETDOWN if *pCookie* does not represent a loaded driver, or an error value returned from the driver's registered *endPollReceive()* function.

ERRNO S_muxLib_NO_DEVICE

SEE ALSO muxLib

muxPollSend()

NAME *muxPollSend()* – send a packet on a network interface

SYNOPSIS

```
STATUS muxPollSend
(
    void*   pCookie, /* cookie the protocol got from muxBind() */
    M_BLK_ID pNBuf   /* data to be sent */
)
```

DESCRIPTION This routine takes a cookie which was returned by *muxBind()* and uses it to determine which network interface driver should be used in transmitting the data. The routine takes the data pointed to by *pNBuf* and sends it to the destination specified by calling the functions in that driver.

pCookie

Expects the cookie returned from *muxBind()*. This Cookie identifies the device to which the MUX has bound the protocol calling *muxPollSend()*.

pNBuf

Expects a pointer to the buffer(**mBlk**) chain that contains the packet to be transmitted.

RETURNS OK, ENETDOWN if *pCookie* doesn't represent a valid device, or ERROR if the device type is not recognized or if the *endPollSend()* routine for the driver fails.

ERRNO S_muxLib_NO_DEVICE

SEE ALSO *muxLib*

muxSend()

NAME *muxSend()* – send a packet out on a network interface

SYNOPSIS

```
STATUS muxSend
(
    void*   pCookie, /* cookie that identifies a network interface */
                    /* by muxBind() */
    M_BLK_ID pNBuf   /* data to be sent */
)
```

| | |
|--------------------|---|
| DESCRIPTION | <p>This routine uses the <i>pCookie</i> value returned during the bind to identify the network interface through which the packet is to be transmitted.</p> <p><i>pCookie</i> Expects the pointer returned from <i>muxBind()</i>. This pointer identifies the device to which the MUX has bound this protocol.</p> <p><i>pNBuf</i> Expects a pointer to the buffer that contains the packet you want to transmit. Before you call <i>muxSend()</i>, you need to put the addressing information at the head of the buffer. To do this, call <i>muxAddressForm()</i>.</p> <p>Also, the buffer should probably be reserved from the MUX- managed memory pool. To reserve a buffer from this pool, the protocol should call <i>muxBufAlloc()</i>.</p> |
| RETURNS | OK, ENETDOWN if <i>pCookie</i> does not represent a valid interface, or ERROR if the driver's <i>endSend()</i> routine fails. |
| ERRNO | S_muxLib_NO_DEVICE |
| SEE ALSO | muxLib |

muxShow()

| | |
|--------------------|--|
| NAME | <i>muxShow()</i> – all configured Enhanced Network Drivers |
| SYNOPSIS | <pre>void muxShow (char * pDevName, /* pointer to device name */ int unit /* unit number for the device */)</pre> |
| DESCRIPTION | <p>If a driver is specified <i>pDevName</i> and <i>unit</i>, this routine reports the name and type of each protocol bound to it. If a <i>pDevName</i> is not given, the entire list of devices and their protocols is shown.</p> <p><i>pDevName</i> Expects a pointer to a string containing the device name, or NULL</p> <p><i>unit</i> Expects a unit number for the device</p> |
| RETURNS | N/A |
| SEE ALSO | muxLib |

muxUnbind()

NAME *muxUnbind()* – detach a protocol from the specified driver

SYNOPSIS

```
STATUS muxUnbind
(
    void*  pCookie,    /* pointer to identifier for device */
    long   type,       /* device type passed in muxBind() call */
    FUNCPTR stackRcvRtn /* pointer to stack receive routine */
)
```

DESCRIPTION

This routine disconnects a protocol from the specified driver.

pCookie
Expects the pointer returned as the function value from the *muxBind()* call. This pointer identifies the device to which the MUX has bound this protocol.

type
This is the type that you passed down in the *muxBind()* call.

stackRcvRtn
Expects a pointer to the stack receive routine you specified when you called *muxBind()* to bind the driver and protocol.

RETURNS OK, EINVAL if *pCookie* does not represent a valid driver or the protocol is not attached, ERROR if *muxUnbind()* fails.

ERRNO EINVAL, S_muxLib_NO_DEVICE

SEE ALSO muxLib

nanosleep()

NAME *nanosleep()* – suspend the current task until the time interval elapses (POSIX)

SYNOPSIS

```
int nanosleep
(
    const struct timespec * rqt, /* time to delay */
    struct timespec *      rmt, /* premature wakeup (NULL=no result) */
)
```


| | |
|--------------------|---|
| DESCRIPTION | <p>This routine suspends the current task for a specified time <i>rqtpor</i> until a signal or event notification is made.</p> <p>The suspension may be longer than requested due to the rounding up of the request to the timer's resolution or to other scheduling activities (e.g., a higher priority task intervenes).</p> <p>If <i>rmtpt</i> is non-NULL, the timespec structure is updated to contain the amount of time remaining. If <i>rmtpt</i> is NULL, the remaining time is not returned. The <i>rqtp</i> parameter is greater than 0 or less than or equal to 1,000,000,000.</p> |
| RETURNS | 0 (OK), or -1 (ERROR) if the routine is interrupted by a signal or an asynchronous event notification, or <i>rqtp</i> is invalid. |
| ERRNO | EINVAL, EINTR |
| SEE ALSO | timerLib , <i>taskDelay()</i> |

ncr710CtrlCreate()

| | |
|--------------------|--|
| NAME | <i>ncr710CtrlCreate()</i> – create a control structure for an NCR 53C710 SIOP |
| SYNOPSIS | <pre> NCR_710_SCSI_CTRL *ncr710CtrlCreate (UINT8 * baseAdrs, /* base address of the SIOP */ UINT freqValue /* clock controller period (nsec* 100) */) </pre> |
| DESCRIPTION | <p>This routine creates an SIOP data structure and must be called before using an SIOP chip. It should be called once and only once for a specified SIOP. Since it allocates memory for a structure needed by all routines in ncr710Lib, it must be called before any other routines in the library. After calling this routine, <i>ncr710CtrlInit()</i> should be called at least once before any SCSI transactions are initiated using the SIOP.</p> <p>A detailed description of the input parameters follows:</p> <p><i>baseAdrs</i> the address at which the CPU accesses the lowest register of the SIOP.</p> <p><i>freqValue</i> the value at the SIOP SCSI clock input. This is used to determine the clock period for the SCSI core of the chip and the synchronous divider value for synchronous transfer. It is important to have the right timing on the SCSI bus. The <i>freqValue</i> parameter is defined as the SCSI clock input value, in nanoseconds, multiplied by 100. Several <i>freqValue</i> constants are defined in ncr710.h as follows:</p> |

```
NCR710_1667MHZ  5998  /* 16.67Mhz chip */
NCR710_20MHZ    5000  /* 20Mhz chip   */
NCR710_25MHZ    4000  /* 25Mhz chip   */
NCR710_3750MHZ  2666  /* 37.50Mhz chip */
NCR710_40MHZ    2500  /* 40Mhz chip   */
NCR710_50MHZ    2000  /* 50Mhz chip   */
NCR710_66MHZ    1515  /* 66Mhz chip   */
NCR710_6666MHZ  1500  /* 66.66Mhz chip */
```

RETURNS A pointer to the `NCR_710_SCSI_CTRL` structure, or `NULL` if memory is insufficient or parameters are invalid.

SEE ALSO `ncr710Lib`

ncr710CtrlCreateScsi2()

NAME `ncr710CtrlCreateScsi2()` – create a control structure for the NCR 53C710 SIOP

SYNOPSIS

```
NCR_710_SCSI_CTRL *ncr710CtrlCreateScsi2
(
    UINT8 * baseAdrs, /* base address of the SIOP */
    UINT   clkPeriod /* clock controller period (nsec* 100) */
)
```

DESCRIPTION This routine creates an SIOP data structure and must be called before using an SIOP chip. It must be called exactly once for a specified SIOP controller. Since it allocates memory for a structure needed by all routines in `ncr710Lib`, it must be called before any other routines in the library. After calling this routine, `ncr710CtrlInitScsi2()` must be called at least once before any SCSI transactions are initiated using the SIOP.

A detailed description of the input parameters follows:

baseAdrs

the address at which the CPU accesses the lowest (SCNTL0/SIEN) register of the SIOP.

clkPeriod

the period of the SIOP SCSI clock input, in nanoseconds, multiplied by 100. This is used to determine the clock period for the SCSI core of the chip and affects the timing of both asynchronous and synchronous transfers. Several commonly used values are defined in `ncr710.h` as follows:

```
NCR710_1667MHZ  6000  /* 16.67Mhz chip */
NCR710_20MHZ    5000  /* 20Mhz chip   */
```

```

NCR710_25MHZ    4000    /* 25Mhz chip */
NCR710_3750MHZ 2667    /* 37.50Mhz chip */
NCR710_40MHZ   2500    /* 40Mhz chip */
NCR710_50MHZ   2000    /* 50Mhz chip */
NCR710_66MHZ   1515    /* 66Mhz chip */
NCR710_6666MHZ 1500    /* 66.66Mhz chip */

```

RETURNS A pointer to the `NCR_710_SCSI_CTRL` structure, or `NULL` if memory is unavailable or there are invalid parameters.

SEE ALSO `ncr710Lib2`

ncr710CtrlInit()

NAME *ncr710CtrlInit()* – initialize a control structure for an NCR 53C710 SIOP

SYNOPSIS

```

STATUS ncr710CtrlInit
(
    NCR_710_SCSI_CTRL * pSiop,          /* ptr to SIOP struct */
    int                scsiCtrlBusId, /* SCSI bus ID of this SIOP */
    int                scsiPriority    /* priority of task when doing SCSI */
)

```

DESCRIPTION This routine initializes an SIOP structure, after the structure is created with *ncr710CtrlCreate()*. This structure must be initialized before the SIOP can be used. It may be called more than once; however, it should be called only while there is no activity on the SCSI interface.

Before returning, this routine pulses RST (reset) on the SCSI bus, thus resetting all attached devices.

The input parameters are as follows:

pSiop

a pointer to the `NCR_710_SCSI_CTRL` structure created with *ncr710CtrlCreate()*.

scsiCtrlBusId

the SCSI bus ID of the SIOP, in the range 0 – 7. The ID is somewhat arbitrary; the value 7, or highest priority, is conventional.

scsiPriority

the priority to which a task is set when performing a SCSI transaction. Valid priorities are 0 to 255. Alternatively, the value -1 specifies that the priority should not be altered during SCSI transactions.

RETURNS OK, or ERROR if parameters are out of range.

SEE ALSO ***ncr710Lib***

ncr710CtrlInitScsi2()

NAME ***ncr710CtrlInitScsi2()*** – initialize a control structure for the NCR 53C710 SIOP

SYNOPSIS

```
STATUS ncr710CtrlInitScsi2
(
    NCR_710_SCSI_CTRL * pSiop,          /* ptr to SIOP struct */
    int                scsiCtrlBusId, /* SCSI bus ID of this SIOP */
    int                scsiPriority /* task priority when doing SCSI I/O */
)
```

DESCRIPTION This routine initializes an SIOP structure after the structure is created with ***ncr710CtrlCreateScsi2()***. This structure must be initialized before the SIOP can be used. It may be called more than once if needed; however, it must only be called while there is no activity on the SCSI interface.

A detailed description of the input parameters follows:

pSiop

a pointer to the NCR_710_SCSI_CTRL structure created with ***ncr710CtrlCreateScsi2()***.

scsiCtrlBusId

the SCSI bus ID of the SIOP. Its value is somewhat arbitrary: seven (7), or highest priority, is conventional. The value must be in the range 0 – 7.

scsiPriority

this parameter is ignored. All SCSI I/O is now done in the context of the SCSI manager task; if necessary, the priority of the manager task may be changed using ***taskPrioritySet()*** or by setting the value of the global variable ***ncr710ScsiTaskPriority*** before calling ***ncr710CtrlCreateScsi2()***.

RETURNS OK, or ERROR if the parameters are out of range.

SEE ALSO ***ncr710Lib2***, ***ncr710CtrlCreateScsi2()***

ncr710SetHwRegister()

NAME *ncr710SetHwRegister()* – set hardware-dependent registers for the NCR 53C710 SIOP

SYNOPSIS

```
STATUS ncr710SetHwRegister
(
    SIOP *          pSiop, /* pointer to SIOP info */
    NCR710_HW_REGS * pHwRegs /* pointer to NCR710_HW_REGS info */
)
```

DESCRIPTION This routine sets up the registers used in the hardware implementation of the chip. Typically, this routine is called by the *sysScsiInit()* routine from the board support package.

The input parameters are as follows:

pSiop

a pointer to the `NCR_710_SCSI_CTRL` structure created with *ncr710CtrlCreate()*.

pHwRegs

a pointer to a `NCR710_HW_REGS` structure that is filled with the logical values 0 or 1 for each bit of each register described below.

This routine includes only the bit registers that can be used to modify the behavior of the chip. The default configuration used during *ncr710CtrlCreate()* and *ncr710CtrlInit()* is {0,0,0,0,1,0,0,0,0,0,0,0,0,1,0}.

```
typedef struct
{
    int ctest4Bit7; /* host bus multiplex mode */
    int ctest7Bit7; /* disable/enable burst cache capability */
    int ctest7Bit6; /* snoop control bit1 */
    int ctest7Bit5; /* snoop control bit0 */
    int ctest7Bit1; /* invert ttl pin (sync bus host mode only) */
    int ctest7Bit0; /* enable differential SCSI bus capability */
    int ctest8Bit0; /* set snoop pins mode */
    int dmodeBit7; /* burst length transfer bit 1 */
    int dmodeBit6; /* burst length transfer bit 0 */
    int dmodeBit5; /* function code bit FC2 */
    int dmodeBit4; /* function code bit FC1 */
    int dmodeBit3; /* program data bit (FC0) */
    int dmodeBit1; /* user-programmable transfer type */
    int dcntlBit5; /* enable ACK pin */
    int dcntlBit1; /* enable fast arbitration on host port */
} NCR710_HW_REGS;
```

For a more detailed description of the register bits, see the *NCR 53C710 SCSI I/O Processor Programming Guide*.

- NOTE** Because this routine writes to the NCR 53C710 chip registers, it cannot be used when there is any SCSI bus activity.
- RETURNS** OK, or ERROR if an input parameter is NULL.
- SEE ALSO** *ncr710Lib*, *ncr710CtrlCreate()*, *NCR 53C710 SCSI I/O Processor Programming Guide*

ncr710SetHwRegisterScsi2()

NAME *ncr710SetHwRegisterScsi2()* – set hardware-dependent registers for the NCR 53C710

SYNOPSIS

```
STATUS ncr710SetHwRegisterScsi2
(
    SIOP *          pSiop, /* pointer to SIOP info */
    NCR710_HW_REGS * pHwRegs /* pointer to a NCR710_HW_REGS info */
)
```

DESCRIPTION This routine sets up the registers used in the hardware implementation of the chip. Typically, this routine is called by the *sysScsiInit()* routine from the BSP.

The input parameters are as follows:

pSiop
a pointer to the *NCR_710 SCSI_CTRL* structure created with *ncr710CtrlCreateScsi2()*.

pHwRegs
a pointer to a *NCR710_HW_REGS* structure that is filled with the logical values 0 or 1 for each bit of each register described below.

This routine includes only the bit registers that can be used to modify the behavior of the chip. The default configuration used during *ncr710CtrlCreateScsi2()* and *ncr710CtrlInitScsi2()* is {0,0,0,0,1,0,0,0,0,0,0,0,0,1,0}.

```
typedef struct
{
    int ctest4Bit7; /* Host bus multiplex mode */
    int ctest7Bit7; /* Disable/enable burst cache capability */
    int ctest7Bit6; /* Snoop control bit1 */
    int ctest7Bit5; /* Snoop control bit0 */
    int ctest7Bit1; /* invert ttl pin (sync bus host mode only)*/
    int ctest7Bit0; /* enable differential scsi bus capability*/
    int ctest8Bit0; /* Set snoop pins mode */
}
```

```

int dmodeBit7;    /* Burst Length transfer bit 1 */
int dmodeBit6;    /* Burst Length transfer bit 0 */
int dmodeBit5;    /* Function code bit FC2 */
int dmodeBit4;    /* Function code bit FC1 */
int dmodeBit3;    /* Program data bit (FC0) */
int dmodeBit1;    /* user programmable transfer type */
int dcntlBit5;    /* Enable Ack pin */
int dcntlBit1;    /* Enable fast arbitration on host port */
} NCR710_HW_REGS;

```

For a more detailed explanation of the register bits, refer to the *NCR 53C710 SCSI I/O Processor Programming Guide*.

- NOTE** Because this routine writes to the chip registers you cannot use it if there is any SCSI bus activity.
- RETURNS** OK, or ERROR if any input parameter is NULL.
- SEE ALSO** *ncr710Lib2*, *ncr710CtrlCreateScsi2()*, *NCR 53C710 SCSI I/O Processor Programming Guide*

ncr710Show()

- NAME** *ncr710Show()* – display the values of all readable NCR 53C710 SIOP registers
- SYNOPSIS**
- ```

STATUS ncr710Show
(
 SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */
)

```
- DESCRIPTION** This routine displays the state of the NCR 53C710 SIOP registers in a user-friendly manner. It is useful primarily for debugging. The input parameter is the pointer to the SIOP information structure returned by the *ncr710CtrlCreate()* call.
- NOTE** The only readable register during a script execution is the Istat register. If this routine is used during the execution of a SCSI command, the result could be unpredictable.
- EXAMPLE**
- ```

-> ncr710Show
NCR710 Registers
-----
0xffff47000: Sien      = 0xa5 Sdid      = 0x00 Scntl11   = 0x00 Scntl10   = 0x04
0xffff47004: Socl      = 0x00 Sodl      = 0x00 Sxfer     = 0x80 Scid       = 0x80
0xffff47008: Sbcl      = 0x00 Sbd1      = 0x00 Sid1      = 0x00 Sfbr      = 0x00
0xffff4700c: Sstat2     = 0x00 Sstat1    = 0x00 Sstat0    = 0x00 Dstat      = 0x80

```

```
0xffff47010: Dsa      = 0x00000000
0xffff47014: Ctest3   = ????. Ctest2   = 0x21 Ctest1   = 0xf0 Ctest0   = 0x00
0xffff47018: Ctest7   = 0x32 Ctest6   = ????. Ctest5   = 0x00 Ctest4   = 0x00
0xffff4701c: Temp     = 0x00000000
0xffff47020: Lcrc    = 0x00 Ctest8   = 0x00 Istat    = 0x00 Dfifo    = 0x00
0xffff47024: Dcmd/Ddc= 0x50000000
0xffff47028: Dnad    = 0x00066144
0xffff4702c: Dsp     = 0x00066144
0xffff47030: Dsps    = 0x00066174
0xffff47037: Scratch3= 0x00 Scratch2= 0x00 Scratch1= 0x00 Scratch0= 0x0a
0xffff47038: Dcntl   = 0x21 Dwt     = 0x00 Dien    = 0x37 Dmode    = 0x01
0xffff4703c: Adder   = 0x00cc2b8
```

RETURNS

OK, or ERROR if *pScsiCtrl* and *pSysScsiCtrl* are both NULL.

SEE ALSO

ncr710Lib, *ncr710CtrlCreate()*

ncr710ShowScsi2()

NAME

ncr710ShowScsi2() – display the values of all readable NCR 53C710 SIOP registers

SYNOPSIS

```
STATUS ncr710ShowScsi2
(
    SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */
)
```

DESCRIPTION

This routine displays the state of the NCR 53C710 SIOP registers in a user-friendly way. It is primarily used for debugging. The input parameter is the pointer to the SIOP information structure returned by the *ncr710CtrlCreateScsi2()* call.

NOTE

The only readable register during a script execution is the Istat register. If you use this routine during the execution of a SCSI command, the result could be unpredictable.

EXAMPLE

```
-> ncr710Show
NCR710 Registers
-----
0xffff47000: Sien    = 0xa5 Sdid    = 0x00 Scntl1   = 0x00 Scntl0   = 0x04
0xffff47004: Socl    = 0x00 Sodl    = 0x00 Sxfer    = 0x80 Scid     = 0x80
0xffff47008: Sbcl    = 0x00 Sbdl    = 0x00 Sidl    = 0x00 Sfbr    = 0x00
0xffff4700c: Sstat2  = 0x00 Sstat1  = 0x00 Sstat0  = 0x00 Dstat    = 0x80
0xffff47010: Dsa     = 0x00000000
0xffff47014: Ctest3  = ????. Ctest2  = 0x21 Ctest1  = 0xf0 Ctest0  = 0x00
```



```

0xffff47018: Ctest7 = 0x32 Ctest6 = ???? Ctest5 = 0x00 Ctest4 = 0x00
0xffff4701c: Temp = 0x00000000
0xffff47020: Lcrc = 0x00 Ctest8 = 0x00 Istat = 0x00 Dfifo = 0x00
0xffff47024: Dcmd/Ddc= 0x50000000
0xffff47028: Dnad = 0x00066144
0xffff4702c: Dsp = 0x00066144
0xffff47030: Dsps = 0x00066174
0xffff47037: Scratch3= 0x00 Scratch2= 0x00 Scratch1= 0x00 Scratch0= 0x0a
0xffff47038: Dcntl = 0x21 Dwt = 0x00 Dien = 0x37 Dmode = 0x01
0xffff4703c: Adder = 0x000cc2b8
value = 0 = 0x0

```

RETURNS OK, or ERROR if *pScsiCtrl* and *pSysScsiCtrl* are both NULL.

SEE ALSO *ncr710Lib2*, *ncr710CtrlCreateScsi2()*

ncr810CtrlCreate()

NAME *ncr810CtrlCreate()* – create a control structure for the NCR 53C8xx SIOP

SYNOPSIS

```

NCR_810_SCSI_CTRL *ncr810CtrlCreate
(
    UINT8 * baseAdrs, /* base address of the SIOP */
    UINT   clkPeriod, /* clock controller period (nsec* 100) */
    UINT16 devType    /* NCR8XX SCSI device type */
)

```

DESCRIPTION This routine creates an SIOP data structure and must be called before using an SIOP chip. It must be called exactly once for a specified SIOP controller. Since it allocates memory for a structure needed by all routines in **ncr810Lib**, it must be called before any other routines in the library. After calling this routine, *ncr810CtrlInit()* must be called at least once before any SCSI transactions are initiated using the SIOP.

A detailed description of the input parameters follows:

baseAdrs
the address at which the CPU accesses the lowest (SCNTL0/SIEN) register of the SIOP.

clkPeriod
the period of the SIOP SCSI clock input, in nanoseconds, multiplied by 100. This is used to determine the clock period for the SCSI core of the chip and affects the timing of both asynchronous and synchronous transfers. Several commonly-used values are defined in **ncr810.h** as follows:

ncr810CtrlInit()

```

NCR810_1667MHZ  6000  /* 16.67Mhz chip */
NCR810_20MHZ    5000  /* 20Mhz chip   */
NCR810_25MHZ    4000  /* 25Mhz chip   */
NCR810_3750MHZ  2667  /* 37.50Mhz chip */
NCR810_40MHZ    2500  /* 40Mhz chip   */
NCR810_50MHZ    2000  /* 50Mhz chip   */
NCR810_66MHZ    1515  /* 66Mhz chip   */
NCR810_6666MHZ  1500  /* 66.66Mhz chip */

```

devType

the specific NCR 8xx device type. Current device types are defined in the header file `ncr810.h`.

RETURNS A pointer to the `NCR_810_SCSI_CTRL` structure, or NULL if memory is unavailable or there are invalid parameters.

SEE ALSO `ncr810Lib`

ncr810CtrlInit()

NAME `ncr810CtrlInit()` – initialize a control structure for the NCR 53C8xx SIOP

SYNOPSIS `STATUS ncr810CtrlInit`

```

(
  NCR_810_SCSI_CTRL * pSiop,          /* ptr to SIOP struct */
  int                scsiCtrlBusId /* SCSI bus ID of this SIOP */
)

```

DESCRIPTION This routine initializes an SIOP structure, after the structure is created with `ncr810CtrlCreate()`. This structure must be initialized before the SIOP can be used. It may be called more than once if needed; however, it must only be called while there is no activity on the SCSI interface. A detailed description of the input parameters follows:

pSiop

a pointer to the `NCR_810_SCSI_CTRL` structure created with `ncr810CtrlCreate()`.

scsiCtrlBusId

the SCSI bus ID of the SIOP. Its value is somewhat arbitrary: seven (7), or highest priority, is conventional. The value must be in the range 0 – 7.

RETURNS OK, or ERROR if parameters are out of range.

SEE ALSO `ncr810Lib`

ncr810SetHwRegister()

NAME *ncr810SetHwRegister()* – set hardware-dependent registers for the NCR 53C8xx SIOP

SYNOPSIS

```
STATUS ncr810SetHwRegister
(
    SIOP *          pSiop, /* pointer to SIOP info */
    NCR810_HW_REGS * pHwRegs /* pointer to a NCR810_HW_REGS info */
)
```

DESCRIPTION This routine sets up the registers used in the hardware implementation of the chip. Typically, this routine is called by the *sysScsiInit()* routine from the BSP.

The input parameters are as follows:

pSiop

a pointer to the NCR_810 SCSI_CTRL structure created with *ncr810CtrlCreate()*.

pHwRegs

a pointer to a NCR810_HW_REGS structure that is filled with the logical values 0 or 1 for each bit of each register described below.

This routine includes only the bit registers that can be used to modify the behavior of the chip. The default configuration used during *ncr810CtrlCreate()* and *ncr810CtrlInit()* is {0,0,0,0,0,1,0,0,0,0,0}.

```
typedef struct
{
    int stest1Bit7;          /* Disable external SCSI clock */
    int stest2Bit7;          /* SCSI control enable */
    int stest2Bit5;          /* Enable differential SCSI bus */
    int stest2Bit2;          /* Always WIDE SCSI */
    int stest2Bit1;          /* Extend SREQ/SACK filtering */
    int stest3Bit7;          /* TolerANT enable */
    int dmodeBit7;           /* Burst Length transfer bit 1 */
    int dmodeBit6;           /* Burst Length transfer bit 0 */
    int dmodeBit5;           /* Source I/O memory enable */
    int dmodeBit4;           /* Destination I/O memory enable*/
    int scntl1Bit7;          /* Slow cable mode */
} NCR810_HW_REGS;
```

For a more detail on the register bits, see the appropriate NCR 53C8xx data manuals.

NOTE Because this routine writes to the NCR 53C8xx chip registers, it cannot be used when there is any SCSI bus activity.

RETURNS OK, or ERROR if any input parameter is NULL

SEE ALSO `ncr810Lib`, `ncr810.h`, `ncr810CtrlCreate()`

ncr810Show()

NAME `ncr810Show()` – display values of all readable NCR 53C8xx SIOP registers

SYNOPSIS

```
STATUS ncr810Show
(
    SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */
)
```

DESCRIPTION This routine displays the state of the SIOP registers in a user-friendly way. It is useful primarily for debugging. The input parameter is the pointer to the SIOP information structure returned by the `ncr810CtrlCreate()` call.

NOTE The only readable register during a script execution is the `Istat` register. If you use this routine during the execution of a SCSI command, the result could be unpredictable.

EXAMPLE

```
-> ncr810Show
NCR810 Registers
-----
0xffff47000: Sien    = 0xa5 Sdid    = 0x00 Scnt11  = 0x00 Scnt10  = 0x04
0xffff47004: Socl    = 0x00 Sodl    = 0x00 Sxfer   = 0x80 Scid    = 0x80
0xffff47008: Sbcl    = 0x00 Sbd1    = 0x00 Sid1    = 0x00 Sfbr   = 0x00
0xffff4700c: Sstat2  = 0x00 Sstat1  = 0x00 Sstat0  = 0x00 Dstat   = 0x80
0xffff47010: Dsa     = 0x00000000
0xffff47014: Ctest3  = ??? Ctest2  = 0x21 Ctest1  = 0xf0 Ctest0  = 0x00
0xffff47018: Ctest7  = 0x32 Ctest6  = ??? Ctest5  = 0x00 Ctest4  = 0x00
0xffff4701c: Temp    = 0x00000000
0xffff47020: Lcrc    = 0x00 Ctest8  = 0x00 Istat   = 0x00 Dfifo   = 0x00
0xffff47024: Dcmd/Ddc= 0x50000000
0xffff47028: Dnad    = 0x00066144
0xffff4702c: Dsp     = 0x00066144
0xffff47030: Dsps    = 0x00066174
0xffff47037: Scratch3= 0x00 Scratch2= 0x00 Scratch1= 0x00 Scratch0= 0x0a
0xffff47038: Dcnt1   = 0x21 Dwt    = 0x00 Dien   = 0x37 Dmode   = 0x01
0xffff4703c: Adder   = 0x000cc2b8
value = 0 = 0x0
```

RETURNS OK, or ERROR if `pScsiCtrl` and `pSysScsiCtrl` are both NULL.

SEE ALSO `ncr810Lib`, `ncr810CtrlCreate()`

ncr5390CtrlCreate()

NAME *ncr5390CtrlCreate()* – create a control structure for an NCR 53C90 ASC

SYNOPSIS

```
NCR_5390_SCSI_CTRL *ncr5390CtrlCreate
(
    UINT8 * baseAdrs,      /* base address of ASC */
    int    regOffset,     /* addr offset between consecutive regs. */
    UINT   clkPeriod,     /* period of controller clock (nsec) */
    FUNCPTR ascDmaBytesIn, /* SCSI DMA input function */
    FUNCPTR ascDmaBytesOut /* SCSI DMA output function */
)
```

DESCRIPTION This routine creates a data structure that must exist before the ASC chip can be used. This routine must be called exactly once for a specified ASC, and must be the first routine called, since it `calloc`'s a structure needed by all other routines in the library.

The input parameters are as follows:

baseAdrs

the address at which the CPU would access the lowest register of the ASC.

regOffset

the address offset (bytes) to access consecutive registers. (This must be a power of 2, for example, 1, 2, 4, etc.)

clkPeriod

the period, in nanoseconds, of the signal to the ASC clock input (used only for select command timeouts).

ascDmaBytesIn and *ascDmaBytesOut*

board-specific parameters to handle DMA input and output. If these are NULL (0), ASC program transfer mode is used. DMA is possible only during SCSI data in/out phases. The interface to these DMA routines must be of the form:

```
STATUS xxDmaBytes{In, Out}
(
    SCSI_PHYS_DEV *pScsiPhysDev, /* ptr to phys dev info */
    UINT8         *pBuffer,      /* ptr to the data buffer */
    int           bufLength      /* number of bytes to xfer */
)
```

RETURNS A pointer to an `NCR_5390_SCSI_CTRL` structure, or NULL if memory is insufficient or the parameters are invalid.

SEE ALSO `ncr5390Lib1`

ncr5390CtrlCreateScsi2()

NAME *ncr5390CtrlCreateScsi2()* – create a control structure for an NCR 53C90 ASC

SYNOPSIS

```
NCR_5390_SCSI_CTRL *ncr5390CtrlCreateScsi2
(
    UINT8* baseAdrs,           /* base address of ASC */
    int    regOffset,         /* offset between consecutive regs. */
    UINT   clkPeriod,        /* period of controller clock (nsec) */
    UINT   sysScsiDmaMaxBytes, /* maximum byte count using DMA */
    FUNCPTR sysScsiDmaStart,  /* function to start SCSI DMA xfer */
    FUNCPTR sysScsiDmaAbort,  /* function to abort SCSI DMA xfer */
    int    sysScsiDmaArg     /* argument to pass to above funcs. */
)
```

DESCRIPTION This routine creates a data structure that must exist before the ASC chip can be used. This routine must be called exactly once for a specified ASC, and must be the first routine called, since it `calloc`'s a structure needed by all other routines in the library.

The input parameters are as follows:

baseAdrs

the address at which the CPU would access the lowest register of the ASC.

regOffset

the address offset (bytes) to access consecutive registers.

clkPeriod

the period, in nanoseconds, of the signal to the ASC clock input.

sysScsiDmaMaxBytes, *sysScsiDmaStart*, *sysScsiDmaAbort*, and *sysScsiDmaArg*

board-specific routines to handle DMA transfers to and from the ASC; if the maximum DMA byte count is zero, programmed I/O is used. Otherwise, non-NULL function pointers to DMA start and abort routines must be provided. The specified argument is passed to these routines when they are called; it may be used to identify the DMA channel to use, for example. The interface to these DMA routines must be of the form:

```
STATUS xxDmaStart (arg, pBuffer, bufLength, direction)
    int arg;           /* call-back argument */
    UINT8 *pBuffer;   /* ptr to the data buffer */
    UINT bufLength;   /* number of bytes to xfer */
    int direction;    /* 0 = SCSI->mem, 1 = mem->SCSI */
STATUS xxDmaAbort (arg)
    int arg;          /* call-back argument */
```

Implementation details for the DMA routines can be found in the specific DMA driver for that board.

- NOTE** If there is no DMA interface, synchronous transfers are not supported. This is a limitation of the NCR5390 hardware.
- RETURNS** A pointer to an `NCR_5390_SCSI_CTRL` structure, or `NULL` if memory is insufficient or the parameters are invalid.
- SEE ALSO** `ncr5390Lib2`

ncr5390CtrlInit()

NAME *ncr5390CtrlInit()* – initialize the user-specified fields in an ASC structure

SYNOPSIS

```
STATUS ncr5390CtrlInit
(
    int * pAsc,           /* ptr to ASC info */
    int  scsiCtrlBusId,  /* SCSI bus ID of this ASC */
    UINT defaultSelTimeOut, /* default dev. select timeout (microsec) */
    int  scsiPriority    /* priority of task when doing SCSI I/O */
)
```

DESCRIPTION This routine initializes an ASC structure, after the structure is created with *ncr5390CtrlCreate()*. This structure must be initialized before the ASC can be used. It may be called more than once; however, it should be called only while there is no activity on the SCSI interface.

Before returning, this routine pulses RST (reset) on the SCSI bus, thus resetting all attached devices. The input parameters are:

pAsc

a pointer to the `NCR5390_SCSI_CTRL` structure created with *ncr5390CtrlCreate()*.

scsiCtrlBusId

the SCSI bus ID of the ASC, in the range 0 – 7. The ID is somewhat arbitrary; the value 7, or highest priority, is conventional.

defaultSelTimeOut

the timeout, in microseconds, for selecting a SCSI device attached to this controller. This value is used as a default if no timeout is specified in *scsiPhysDevCreate()*. The recommended value zero (0) specifies `SCSI_DEF_SELECT_TIMEOUT` (250 millisecc). The maximum timeout possible is approximately 2 seconds. Values exceeding this revert to the maximum.

scsiPriority

the priority to which a task is set when performing a SCSI transaction. Valid priorities are 0 to 255. Alternatively, the value -1 specifies that the priority should not be altered during SCSI transactions.

RETURNS OK, or ERROR if a parameter is out of range.

SEE ALSO `ncr5390Lib`, `scsiPhysDevCreate()`,

ncr5390Show()

NAME `ncr5390Show()` – display the values of all readable NCR5390 chip registers

SYNOPSIS

```
int ncr5390Show
(
    int * pScsiCtrl /* ptr to SCSI controller info */
)
```

DESCRIPTION This routine displays the state of the ASC registers in a user-friendly manner. It is useful primarily for debugging. It should not be invoked while another running process is accessing the SCSI controller.

EXAMPLE

```
-> ncr5390Show
REG #00 (Own ID           ) = 0x07
REG #01 (Control         ) = 0x00
REG #02 (Timeout Period ) = 0x20
REG #03 (Sectors         ) = 0x00
REG #04 (Heads           ) = 0x00
REG #05 (Cylinders MSB  ) = 0x00
REG #06 (Cylinders LSB  ) = 0x00
REG #07 (Log. Addr. MSB ) = 0x00
REG #08 (Log. Addr. 2SB ) = 0x00
REG #09 (Log. Addr. 3SB ) = 0x00
REG #0a (Log. Addr. LSB ) = 0x00
REG #0b (Sector Number  ) = 0x00
REG #0c (Head Number    ) = 0x00
REG #0d (Cyl. Number MSB) = 0x00
REG #0e (Cyl. Number LSB) = 0x00
REG #0f (Target LUN     ) = 0x00
REG #10 (Command Phase  ) = 0x00
REG #11 (Synch. Transfer) = 0x00
REG #12 (Xfer Count MSB ) = 0x00
```



```
REG #13 (Xfer Count 2SB ) = 0x00
REG #14 (Xfer Count LSB ) = 0x00
REG #15 (Destination ID ) = 0x03
REG #16 (Source ID      ) = 0x00
REG #17 (SCSI Status    ) = 0x42
REG #18 (Command        ) = 0x07
```

RETURNS OK, or ERROR if *pScsiCtrl* and *pSysScsiCtrl* are both NULL.

SEE ALSO *ncr5390Lib*

ncr710SingleStep()

NAME *ncr710SingleStep()* – perform a single-step

SYNOPSIS

```
void ncr710SingleStep
(
    SIOP * pSiop, /* pointer to SIOP info */
    BOOL  verbose /* show all registers */
)
```

DESCRIPTION This routine performs a single-step by writing the STD bit in the DCNTL register. The *pSiop* parameter is a pointer to the SIOP information. Before executing, enable the single-step facility by calling *ncr710StepEnable()*.

RETURNS N/A

SEE ALSO *ncr710CommLib*, *ncr710StepEnable()*

ncr710StepEnable()

NAME *ncr710StepEnable()* – enable/disable script single-step

SYNOPSIS

```
void ncr710StepEnable
(
    SIOP * pSiop, /* pointer to SIOP info */
    BOOL  boolValue /* TRUE/FALSE to enable/disable single step */
)
```

| | |
|--------------------|--|
| DESCRIPTION | This routine enables/disables the single-step facility on the chip. It also unmask/masks the single-step interrupt in the Dien register. Before executing any SCSI routines, enable the single-step facility by calling <i>ncr710StepEnable()</i> with <i>boolValue</i> set to TRUE. To disable, call it with <i>boolValue</i> set to FALSE. |
| RETURNS | N/A |
| SEE ALSO | <i>ncr710CommLib</i> , <i>ncr710SingleStep()</i> |

ne2000EndLoad()

| | |
|--------------------|--|
| NAME | <i>ne2000EndLoad()</i> – initialize the driver and device |
| SYNOPSIS | <pre>END_OBJ* ne2000EndLoad (char* initString, /* String to be parsed by the driver. */ void* pBSP /* for BSP group */)</pre> |
| DESCRIPTION | <p>This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the <i>initString</i>.</p> <p>The string contains the target specific parameters like this:</p> <p>"unit:register addr:int vector:int level:shmem addr:shmem size:shmem width"</p> |
| RETURNS | An END object pointer or NULL on error. |
| SEE ALSO | <i>ne2000End</i> |

ne2000Parse()

| | |
|-----------------|---|
| NAME | <i>ne2000Parse()</i> – parse the init string |
| SYNOPSIS | <pre>STATUS ne2000Parse (NE2000END_DEVICE * pDrvCtrl, char * initString)</pre> |

DESCRIPTION Parse the input string. Fill in values in the driver control structure.
 The initialization string format is:
unit:adrs:vecnum:intLvl:byteAccess:usePromEnetAddr:offset

unit
 Device unit number, a small integer.

adrs
 Base address

vecNum
 Interrupt vector number (used with sysIntConnect)

intLvl
 Interrupt level (used with sysLanIntEnable)

byteAccess
 Use 8-bit access mode.

usePromEnetAddr
 get ethernet address from PROM.

offset
 offset for memory alignment

RETURNS OK or ERROR for invalid arguments.

SEE ALSO **ne2000End**

netBufLibInit()

NAME *netBufLibInit()* – initialize netBufLib

SYNOPSIS `STATUS netBufLibInit (void)`

DESCRIPTION This routine initializes **netBufLib**. If you defined `INCLUDE_NETWORK` in `configAll.h`, this configured VxWorks to include **netBufLib**.

RETURNS OK or ERROR.

SEE ALSO **netBufLib**

netClBlkFree()

NAME *netClBlkFree()* – free a **clBlk**-cluster construct back to the memory pool

SYNOPSIS

```
void netClBlkFree
(
    NET_POOL_ID pNetPool, /* pointer to the net pool */
    CL_BLK_ID   pClBlk    /* pointer to the clBlk to free */
)
```

DESCRIPTION This routine decrements the reference counter in the specified **clBlk**. If the reference count falls to zero, this routine frees both the **clBlk** and its associated cluster back to the specified memory pool.

RETURNS N/A

SEE ALSO **netBufLib**

netClBlkGet()

NAME *netClBlkGet()* – get a **clBlk**

SYNOPSIS

```
CL_BLK_ID netClBlkGet
(
    NET_POOL_ID pNetPool, /* pointer to the net pool */
    int         canWait   /* M_WAIT/M_DONTWAIT */
)
```

DESCRIPTION This routine gets a **clBlk** from the specified memory pool.

pNetPool

Expects a pointer to the pool from which you want a **clBlk**.

canWait

Expects either **M_WAIT** or **M_DONTWAIT**. If *canWait* is **M_WAIT**, this routine blocks until a **clBlk** is available. If *canWait* is **M_DONTWAIT** and no **clBlk** is immediately available, this routine returns immediately (no blocking) with a NULL value.

RETURNS **CL_BLK_ID** or a NULL if no **clBlk** was available.

SEE ALSO **netBufLib**

netClBlkJoin()

NAME *netClBlkJoin()* – join a cluster to a **clBlk** structure

SYNOPSIS

```
CL_BLK_ID netClBlkJoin
(
    CL_BLK_ID pClBlk, /* pointer to a cluster Blk */
    char *    pClBuf, /* pointer to a cluster buffer */
    int       size,   /* size of the cluster buffer */
    FUNCPTR   pFreeRtn, /* pointer to the free routine */
    int       arg1,   /* argument 1 of the free routine */
    int       arg2,   /* argument 2 of the free routine */
    int       arg3    /* argument 3 of the free routine */
)
```

DESCRIPTION This routine joins the previously reserved cluster specified by *pClBuf* to the previously reserved **clBlk** structure specified by *pClBlk*. The *size* parameter passes in the size of the cluster referenced in *pClBuf*. The arguments *pFreeRtn*, *arg1*, *arg2*, *arg3* set the values of the **pClFreeRtn**, **clFreeArg1**, **clFreeArg2**, and **clFreeArg1**, members of the specified **clBlk** structure.

RETURNS **CL_BLK_ID** or **NULL**.

SEE ALSO **netBufLib**

netClFree()

NAME *netClFree()* – free a cluster back to the memory pool

SYNOPSIS

```
void netClFree
(
    NET_POOL_ID pNetPool, /* pointer to the net pool */
    UCHAR *     pClBuf    /* pointer to the cluster buffer */
)
```

DESCRIPTION This routine returns the specified cluster buffer back to the specified memory pool.

RETURNS N/A

SEE ALSO **netBufLib**

netClPoolIdGet()

NAME *netClPoolIdGet()* – return a CL_POOL_ID for a specified buffer size

SYNOPSIS

```
CL_POOL_ID netClPoolIdGet
(
    NET_POOL_ID pNetPool, /* pointer to the net pool */
    int         bufSize,  /* size of the buffer */
    BOOL        bestFit   /* TRUE/FALSE */
)
```

DESCRIPTION This routine returns a CL_POOL_ID for a cluster pool containing clusters that match the specified *bufSize*. If *bestFit* is TRUE, this routine returns a CL_POOL_ID for a pool that contains clusters greater than or equal to *bufSize*. If *bestFit* is FALSE, this routine returns a CL_POOL_ID for a cluster from whatever cluster pool is available. If the memory pool specified by *pNetPool* contains only one cluster pool, *bestFit* should always be FALSE.

RETURNS CL_POOL_ID or NULL.

SEE ALSO **netBufLib**

netClusterGet()

NAME *netClusterGet()* – get a cluster from the specified cluster pool

SYNOPSIS

```
char * netClusterGet
(
    NET_POOL_ID pNetPool, /* pointer to the net pool */
    CL_POOL_ID  pClPool   /* ptr to the cluster pool */
)
```

DESCRIPTION This routine gets a cluster from the specified cluster pool *pClPool* within the specified memory pool *pNetPool*.

RETURNS This routine returns a character pointer to a cluster buffer or NULL if none was available.

SEE ALSO **netBufLib**

netDevCreate()

NAME *netDevCreate()* – create a remote file device

SYNOPSIS

```
STATUS netDevCreate
(
    char * devName, /* name of device to create */
    char * host,    /* host this device will talk to */
    int   protocol /* remote file access protocol 0 = RSH, 1 = FTP */
)
```

DESCRIPTION This routine creates a remote file device. Normally, a network device is created for each remote machine whose files are to be accessed. By convention, a network device name is the remote machine name followed by a colon ":". For example, for a UNIX host on the network whose name is "wrs", files can be accessed by creating a device called "wrs:". Files can be accessed via RSH as follows:

```
netDevCreate ("wrs:", "wrs", rsh);
```

The file /usr/dog on the UNIX system "wrs" can now be accessed as "wrs:/usr/dog" via RSH.

Before creating a device, the host must have already been created with *hostAdd()*.

RETURNS OK or ERROR.

SEE ALSO *netDrv*, *hostAdd()*

netDrv()

NAME *netDrv()* – install the network remote file driver

SYNOPSIS STATUS *netDrv* (void)

DESCRIPTION This routine initializes and installs the network driver. It must be called before other network remote file functions are performed. It is called automatically when the configuration macro `INCLUDE_NETWORK` is defined.

RETURNS OK or ERROR.

SEE ALSO *netDrv*

netHelp()

NAME *netHelp()* – print a synopsis of network routines

SYNOPSIS `void netHelp (void)`

DESCRIPTION This command prints a brief synopsis of network facilities that are typically called from the shell.

```
hostAdd      "hostname", "inetaddr" - add a host to remote host table;
                                     "inetaddr" must be in standard
                                     Internet address format e.g.
"90.0.0.4"
hostShow      - print current remote host table
netDevCreate "devname", "hostname", protocol
                                     - create an I/O device to access
                                     files on the specified host
                                     (protocol 0=rsh, 1=ftp)
routeAdd     "destaddr", "gateaddr" - add route to route table
routeDelete  "destaddr", "gateaddr" - delete route from route table
routeShow    - print current route table
iam          "usr" [, "passwd"]      - specify the user name by which
                                     you will be known to remote
                                     hosts (and optional password)
whoami       - print the current remote ID
rlogin      "host"                  - log in to a remote host;
                                     "host" can be inet address or
                                     host name in remote host table
ifShow      ["ifname"]              - show info about network interfaces
inetstatShow - show all Internet protocol sockets
tcpstatShow - show statistics for TCP
udpstatShow - show statistics for UDP
ipstatShow  - show statistics for IP
icmpstatShow - show statistics for ICMP
arptabShow  - show a list of known ARP entries
mbufShow    - show mbuf statistics
EXAMPLE:    -> hostAdd "wrs", "90.0.0.2"
            -> netDevCreate "wrs:", "wrs", 0
            -> iam "fred"
            -> copy <wrs:/etc/passwd /* copy file from host "wrs" */
            -> rlogin "wrs" /* rlogin to host "wrs" */
```

RETURNS N/A

SEE ALSO `usrLib`, *VxWorks Programmer's Guide: Target Shell*

netLibInit()

| | |
|--------------------|--|
| NAME | <i>netLibInit()</i> – initialize the network package |
| SYNOPSIS | <code>STATUS netLibInit (void)</code> |
| DESCRIPTION | This creates the network task job queue, and spawns the network task <i>netTask()</i> . It should be called once to initialize the network. This is done automatically when the configuration macro <code>INCLUDE_NETWORK</code> is defined. |
| RETURNS | OK, or ERROR if network support cannot be initialized. |
| SEE ALSO | <i>netLib</i> , <i>usrConfig</i> , <i>netTask()</i> |

netMblkChainDup()

| | |
|--------------------|---|
| NAME | <i>netMblkChainDup()</i> – duplicate an mBlk chain |
| SYNOPSIS | <pre>M_BLK_ID netMblkChainDup (NET_POOL_ID pNetPool, /* pointer to the pool */ M_BLK_ID pMblk, /* pointer to source mBlk chain */ int offset, /* offset to duplicate from */ int len, /* length to copy */ int canWait /* M_DONTWAIT/M_WAIT */)</pre> |
| DESCRIPTION | <p>This routine makes a copy of an mBlk chain starting at <i>offset</i> bytes from the beginning of the chain and continuing for <i>len</i> bytes. If <i>len</i> is <code>M_COPYALL</code>, then this routine will copy the entire mBlk chain from the <i>offset</i>.</p> <p>This routine copies the references from a source <i>pMblk</i> chain to a newly allocated mBlk chain. This lets the two mBlk chains share the same clBlk-cluster constructs. This routine also increments the reference count in the shared clBlk. The <i>pMblk</i> expects a pointer to the source mBlk chain. The <i>pNetPool</i> parameter expects a pointer to the netPool from which the new mBlk chain is allocated.</p> <p>The <i>canWait</i> parameter expects either <code>M_WAIT</code> or <code>M_DONTWAIT</code>. If <i>canWait</i> is <code>M_WAIT</code>, this routine blocks until mBlk is available. If <i>canWait</i> is <code>M_DONTWAIT</code> and no mBlk is immediately available, this routine returns immediately (no blocking) with a NULL value.</p> |

SEE ALSO *netMblkDup()*

RETURNS A pointer to the newly allocated **mBlk** chain or NULL.

ERRNO **S_netBufLib_INVALID_ARGUMENT**
 S_netBufLib_NO_POOL_MEMORY

netMblkClChainFree()

NAME *netMblkClChainFree()* – free a chain of **mBlk-clBlk**-cluster constructs

SYNOPSIS `void netMblkClChainFree`
 (
 M_BLK_ID pMblk /* pointer to the mBlk */
)

DESCRIPTION For the specified chain of **mBlk-clBlk**-cluster constructs, this routine frees all the **mBlk** structures back to the specified memory pool. It also decrements the reference count in all the **clBlk** structures. If the reference count in a **clBlk** falls to zero, this routine also frees that **clBlk** and its associated cluster back to the specified memory pool.

RETURNS N/A

ERRNO **S_netBufLib_MBLK_INVALID**

SEE ALSO **netBufLib**

netMblkClFree()

NAME *netMblkClFree()* – free an **mBlk-clBlk**-cluster construct

SYNOPSIS `M_BLK_ID netMblkClFree`
 (
 M_BLK_ID pMblk /* pointer to the mBlk */
)

DESCRIPTION For the specified **mBlk-clBlk**-cluster construct, this routine frees the **mBlk** back to the specified memory pool. It also decrements the reference count in the **clBlk** structure. If the reference count falls to zero, no other **mBlk** structure reference this **clBlk**. In that case,

this routine also frees the **cBlk** structure and its associated cluster back to the specified memory pool.

| | |
|-----------------|---|
| RETURNS | If the specified mBlk was part of an mBlk chain, this routine returns a pointer to the next mBlk . Otherwise, it returns a NULL. |
| ERRNO | <code>S_netBufLib_MBLK_INVALID</code> |
| SEE ALSO | <code>netBufLib</code> |

netMblkClGet()

NAME *netMblkClGet()* – get a **cBlk**-cluster and join it to the specified **mBlk**

SYNOPSIS

```

STATUS netMblkClGet
(
    NET_POOL_ID pNetPool, /* pointer to the net pool */
    M_BLK_ID    pMblk,    /* mBlk to embed the cluster in */
    int         bufSize,  /* size of the buffer to get */
    int         canWait,  /* wait or dontwait */
    BOOL        bestFit   /* TRUE/FALSE */
)

```

DESCRIPTION This routine gets a **cBlk**-cluster construct from the specified memory pool and joins it to the specified **mBlk** structure. This creates an **mBlk-cBlk**-cluster construct that you can use to pass data across the layers of the network stack.

pNetPool

Expects a pointer to the memory pool from which you want to get a free **cBlk**-cluster construct.

pMblk

Expects a pointer to the **mBlk** structure (previously allocated) to which you want to join the retrieved **cBlk**-cluster construct.

bufSize

Expects the size, in bytes, of the cluster in the **cBlk**-cluster construct.

canWait

Expects either **M_WAIT** or **M_DONTWAIT**. If *canWait* is **M_WAIT**, this routine blocks until a **cBlk**-cluster construct is available. If *canWait* is **M_DONTWAIT** and no **cBlk**-cluster construct is immediately available, this routine returns immediately (no blocking) with an ERROR value.

bestFit

Expects either TRUE or FALSE. If *bestFit* is TRUE and a cluster of the exact size is unavailable, this routine gets a larger cluster (if available). If *bestFit* is FALSE and an exact size cluster is unavailable, this routine gets either a smaller or a larger cluster (depending on what is available). Otherwise, it returns immediately with an ERROR value. For memory pools containing only one cluster size, *bestFit* should always be set to FALSE.

RETURNS OK or ERROR.

ERRNO S_netBufLib_CLSIZE_INVALID

SEE ALSO netBufLib

netMblkClJoin()

NAME *netMblkClJoin()* – join an **mBlk** to a **clBlk**-cluster construct

SYNOPSIS

```
M_BLK_ID netMblkClJoin
(
    M_BLK_ID pMblk, /* pointer to an mBlk */
    CL_BLK_ID pClBlk /* pointer to a cluster Blk */
)
```

DESCRIPTION This routine joins the previously reserved **mBlk** referenced in *pMblk* to the **clBlk**-cluster construct referenced in *pClBlk*. Internally, this routine sets the **M_EXT** flag in **mBlk.mBlkHdr.mFlags**. It also and sets the **mBlk.mBlkHdr.mData** to point to the start of the data in the cluster.

RETURNS M_BLK_ID or NULL.

SEE ALSO netBufLib

netMblkDup()

| | |
|--------------------|---|
| NAME | <i>netMblkDup()</i> – duplicate an mBlk |
| SYNOPSIS | <pre>M_BLK_ID netMblkDup (M_BLK_ID pSrcMblk, /* pointer to source mBlk */ M_BLK_ID pDestMblk /* pointer to the destination mBlk */)</pre> |
| DESCRIPTION | This routine copies the references from a source mBlk in an mBlk-clBlk -cluster construct to a stand-alone mBlk . This lets the two mBlk structures share the same clBlk -cluster construct. This routine also increments the reference count in the shared clBlk . The <i>pSrcMblk</i> expects a pointer to the source mBlk . The <i>pDestMblk</i> parameter expects a pointer to the destination mBlk . |
| RETURNS | A pointer to the destination mBlk or NULL if the source mBlk referenced in <i>pSrcMblk</i> is not part of a valid mBlk-clBlk -cluster construct. |
| SEE ALSO | netBufLib |

netMblkFree()

| | |
|--------------------|---|
| NAME | <i>netMblkFree()</i> – free an mBlk back to its memory pool |
| SYNOPSIS | <pre>void netMblkFree (NET_POOL_ID pNetPool, /* pointer to the net pool */ M_BLK_ID pMblk /* mBlk to free */)</pre> |
| DESCRIPTION | This routine frees the specified mBlk back to the specified memory pool. |
| RETURNS | N/A |
| SEE ALSO | netBufLib |

netMblkGet()

netMblkGet()**NAME** *netMblkGet()* – get an mBlk

SYNOPSIS `M_BLK_ID netMblkGet`

```
(
    NET_POOL_ID pNetPool, /* pointer to the net pool */
    int         canWait,  /* M_WAIT/M_DONTWAIT */
    UCHAR       type     /* mBlk type */
)
```

DESCRIPTION This routine gets a **mBlk** from the specified memory pool.*pNetPool*Expects a pointer to the pool from which you want an **mBlk**.*canWait*Expects either **M_WAIT** or **M_DONTWAIT**. If *canWait* is **M_WAIT**, this routine blocks until an **mBlk** is available. If *canWait* is **M_DONTWAIT** and no **mBlk** is immediately available, this routine returns immediately (no blocking) with a NULL value.*type*Expects the type value that you want to associate with the returned **mBlk**.**RETURNS** **M_BLK_ID**, or a NULL if no **mBlk** was available.**ERRNO** **S_netBufLib_MBLK_INVALID****SEE ALSO** **netBufLib**

netMblkToBufCopy()**NAME** *netMblkToBufCopy()* – copy data from an mBlk to a buffer

SYNOPSIS `int netMblkToBufCopy`

```
(
    M_BLK_ID pMblk, /* pointer to an mBlk */
    char *   pBuf,  /* pointer to the buffer to copy */
    FUNCPTR pCopyRtn /* function pointer for copy routine */
)
```

- DESCRIPTION** This routine copies data from the **mBlk** chain referenced in *pMblk* to the buffer referenced in *pBuf*. It is assumed that *pBuf* points to enough memory to contain all the data in the entire **mBlk** chain. The argument *pCopyRtn* expects either a NULL or a function pointer to a copy routine. The arguments passed to the copy routine are source pointer, destination pointer and the length of data to copy. If *pCopyRtn* is NULL, *netMblkToBufCopy()* uses a default routine to extract the data from the chain.
- RETURNS** The length of data copied or zero.
- SEE ALSO** **netBufLib**

netPoolDelete()

- NAME** *netPoolDelete()* – delete a memory pool
- SYNOPSIS**
- ```
STATUS netPoolDelete
(
 NET_POOL_ID pNetPool /* pointer to a net pool */
)
```
- DESCRIPTION** This routine deletes the specified **netBufLib**-managed memory pool.
- RETURNS** OK or ERROR.
- ERRNO** S\_netBufLib\_NETPOOL\_INVALID
- SEE ALSO** **netBufLib**

## *netPoolInit()*

- NAME** *netPoolInit()* – initialize a **netBufLib**-managed memory pool
- SYNOPSIS**
- ```
STATUS netPoolInit
(
    NET_POOL_ID    pNetPool,          /* pointer to a net pool */
    M_CL_CONFIG * pMclBlkConfig,     /* pointer to a mBlk configuration */
    CL_DESC *     pClDescTbl,        /* pointer to cluster desc table */
    int           clDescTblNumEnt,    /* number of cluster desc entries */
    POOL_FUNC *  pFuncTbl            /* pointer to pool function table */
)
```

netPoolInit()**DESCRIPTION**

Call this routine to set up a **netBufLib**-managed memory pool. Within this pool, **netPoolInit()** organizes several sub-pools: one for **mBlk** structures, one for **cBlk** structures, and as many cluster sub-pools as there are cluster sizes. As input, this routine expects the following parameters:

pNetPool

Expects a **NET_POOL_ID** that points to a previously allocated **NET_POOL** structure. You need not initialize any values in this structure. That is handled by **netPoolInit()**.

pMclBlkConfig

Expects a pointer to a previously allocated and initialized **M_CL_CONFIG** structure. Within this structure, you must provide four values: **mBlkNum**, a count of **mBlk** structures; **cBlkNum**, a count of **cBlk** structures; **memArea**, a pointer to an area of memory that can contain all the **mBlk** and **cBlk** structures; and **memSize**, the size of that memory area. For example, you can set up an **M_CL_CONFIG** structure as follows:

```
M_CL_CONFIG mClBlkConfig = /* mBlk, cBlk configuration table */
{
    mBlkNum      cBlkNum      memArea      memSize
    -----    ----          -
    400,         245,         0xfe000000,    21260
};
```

You can calculate the **memArea** and **memSize** values. Such code could first define a table as shown above, but set both **memArea** and **memSize** as follows:

```
mClBlkConfig.memSize = (mClBlkConfig.mBlkNum * (M_BLK_SZ + sizeof(long))) +
    (mClBlkConfig.cBlkNum * CL_BLK_SZ);
```

You can set the **memArea** value to a pointer to private memory, or you can reserve the memory with a call to **malloc()**. For example:

```
mClBlkConfig.memArea = malloc(mClBlkConfig.memSize);
```

The **netBufLib.h** file defines **M_BLK_SZ** as:

```
sizeof(struct mBlk)
```

Currently, this evaluates to 32 bytes. Likewise, this file defines **CL_BLK_SZ** as:

```
sizeof(struct cBlk)
```

Currently, this evaluates to 32 bytes.

When choosing values for **mBlkNum** and **cBlkNum**, remember that you need as many **cBlk** structures as you have clusters (data buffers). You also need at least as many **mBlk** structures as you have **cBlk** structures, but you will most likely need more. That is because **netBufLib** shares buffers by letting multiple **mBlk** structures join to the same **cBlk** and thus to its underlying cluster. The **cBlk** keeps a count of the number of **mBlk** structures that reference it.

pCIDescTbl

Expects a pointer to a table of previously allocated and initialized **CL_DESC** structures. Each structure in this table describes a single cluster pool. You need a dedicated cluster pool for each cluster size you want to support. Within each **CL_DESC** structure, you must provide four values: **clusterSize**, the size of a cluster in this cluster pool; **num**, the number of clusters in this cluster pool; **memArea**, a pointer to an area of memory that can contain all the clusters; and **memSize**, the size of that memory area.

Thus, if you need to support six different cluster sizes, this parameter must point to a table containing six **CL_DESC** structures. For example, consider the following:

```
CL_DESC clDescTbl [] = /* cluster descriptor table */
{
/*
clusterSize      num      memArea      memSize
-----
*/
{64,             100,    0x10000,    6800},
{128,            50,     0x20000,    6600},
{256,            50,     0x30000,    13000},
{512,            25,     0x40000,    12900},
{1024,           10,     0x50000,    10280},
{2048,           10,     0x60000,    20520}
};
```

As with the **memArea** and **memSize** members in the **M_CL_CONFIG** structure, you can set these members of the **CL_DESC** structures by calculation after you create the table. The formula would be as follows:

```
clDescTbl[n].memSize =
    (clDescTbl[n].num * (clDescTbl[n].clusterSize + sizeof(long)));
```

The **memArea** member can point to a private memory area that you know to be available for storing clusters, or you can use *malloc()*.

```
clDescTbl[n].memArea = malloc( clDescTbl[n].memSize );
```

Valid cluster sizes range from 64 bytes to 65536 bytes. If there are multiple cluster pools, valid sizes are further restricted to powers of two (for example, 64, 128, 256, and so on). If there is only one cluster pool (as is often the case for the memory pool specific to a single device driver), there is no power of two restriction. Thus, the cluster can be of any size between 64 bytes and 65536 bytes on 4-byte alignment. A typical buffer size for Ethernet devices is 1514 bytes. However, because a cluster size requires a 4-byte alignment, the cluster size for this Ethernet buffer would have to be increased to at least 1516 bytes.

clDescTblNumEnt

Expects a count of the elements in the **CL_DESC** table referenced by the *pCIDescTbl* parameter. This is a count of the number of cluster pools. You can get this value

netPoolShow()

using the NELEMENTS macro defined in `vxWorks.h`. For example:

```
int clDescTblNumEnt = (NELEMENTS(clDescTbl));
```

pFuncTbl

Expects a NULL or a pointer to a function table. This table contains pointers to the functions used to manage the buffers in this memory pool. Using a NULL for this parameter tells `netBufLib` to use its default function table. If you opt for the default function table, every `mBlk` and every cluster is prepended by a 4-byte header (which is why the size calculations above for clusters and `mBlk` structures contained an extra `sizeof(long)`). However, users need not concern themselves with this header when accessing these buffers. The returned pointers from functions such as `netClusterGet()` return pointers to the start of data, which is just after the header.

Assuming you have set up the configuration tables as shown above, a typical call to `netPoolInit()` would be as follows:

```
int clDescTblNumEnt = (NELEMENTS(clDescTbl));
NET_POOL netPool;
NET_POOL_ID pNetPool = &netPool;
if (netPoolInit (pNetPool, &mClBlkConfig, &clDescTbl [0],
clDescTblNumEnt,
NULL) != OK)
return (ERROR);
```

RETURNS

OK or ERROR.

ERRNO

S_netBufLib_MEMSIZE_INVALID
S_netBufLib_CLSIZE_INVALID
S_netBufLib_NO_SYSTEM_MEMORY
S_netBufLib_MEM_UNALIGNED
S_netBufLib_MEMSIZE_UNALIGNED
S_netBufLib_MEMAREA_INVALID

SEE ALSO

`netBufLib`, `netPoolDelete()`

netPoolShow()

NAME

`netPoolShow()` – show pool statistics

SYNOPSIS

```
void netPoolShow
(
NET_POOL_ID pNetPool
)
```

DESCRIPTION This routine displays the distribution of **mBlks** and clusters in a given network pool ID.

RETURNS N/A

SEE ALSO **netShow**

netShowInit()

NAME *netShowInit()* – initialize network show routines

SYNOPSIS `void netShowInit (void)`

DESCRIPTION This routine links the network show facility into the VxWorks system. These routines are included automatically if `INCLUDE_NET_SHOW` is defined in `configAll.h`.

RETURNS N/A

SEE ALSO **netShow**

netStackDataPoolShow()

NAME *netStackDataPoolShow()* – show network stack data pool statistics

SYNOPSIS `void netStackDataPoolShow (void)`

DESCRIPTION This routine displays the distribution of **mBlks** and clusters in a the network data pool. The network data pool is used only for data transfer through the network stack.

RETURNS N/A

SEE ALSO **netShow**, *netStackSysPoolShow()*, **netBufLib**

netStackSysPoolShow()

| | |
|--------------------|---|
| NAME | <i>netStackSysPoolShow()</i> – show network stack system pool statistics |
| SYNOPSIS | <code>void netStackSysPoolShow (void)</code> |
| DESCRIPTION | This routine displays the distribution of mBlks and clusters in a the network system pool. The network system pool is used only for system structures such as sockets, routes, interface addresses, protocol control blocks, multicast addresses, and multicast route entries. |
| RETURNS | N/A |
| SEE ALSO | netShow , netStackDataPoolShow() , netBufLib |

netTask()

| | |
|--------------------|--|
| NAME | <i>netTask()</i> – network task entry point |
| SYNOPSIS | <code>void netTask (void)</code> |
| DESCRIPTION | This routine is the VxWorks network support task. Most of the VxWorks network runs in this task's context. |
| NOTE | To prevent an application task from monopolizing the CPU if it is in an infinite loop or is never blocked, the priority of <i>netTask()</i> relative to an application may need to be adjusted. Network communication may be lost if <i>netTask()</i> is "starved" of CPU time. The default task priority of <i>netTask()</i> is 50. Use <i>taskPrioritySet()</i> to change the priority of a task. This task is spawned by <i>netLibInit()</i> . |
| RETURNS | N/A |
| SEE ALSO | netLibInit() |

netTupleGet()

| | |
|--------------------|---|
| NAME | <i>netTupleGet()</i> – get an mBlk-clBlk -cluster |
| SYNOPSIS | <pre>M_BLK_ID netTupleGet (NET_POOL_ID pNetPool, /* pointer to the net pool */ int bufSize, /* size of the buffer to get */ int canWait, /* wait or dontwait */ UCHAR type, /* type of data */ BOOL bestFit /* TRUE/FALSE */)</pre> |
| DESCRIPTION | <p>This routine gets a mBlk-clBlk-cluster construct from the specified memory pool. Use this construct to pass data across the layers of the network stack.</p> <p><i>pNetPool</i> Expects a pointer to the memory pool from which you want to get a free mBlk-clBlk-cluster construct.</p> <p><i>bufSize</i> Expects the size, in bytes, of the cluster in the clBlk-cluster construct.</p> <p><i>canWait</i> Expects either M_WAIT or M_DONTWAIT. If <i>canWait</i> is M_WAIT, this routine blocks until an mBlk-clBlk-cluster construct is available. If <i>canWait</i> is M_DONTWAIT and no mBlk-clBlk-cluster construct is immediately available, this routine returns immediately (no blocking) with a NULL value.</p> <p><i>type</i> Expects the type of data. For example MT_DATA, MT_HEADER. The various values for this type are defined in netBufLib.h.</p> <p><i>bestFit</i> Expects either TRUE or FALSE. If TRUE and a cluster of the exact size is unavailable, this routine gets a larger cluster (if available). If <i>bestFit</i> is FALSE and an exact size cluster is unavailable, this routine gets either a smaller or a larger cluster (depending on what is available). Otherwise, it returns immediately with an ERROR value. For memory pools containing only one cluster size, <i>bestFit</i> should always be set to FALSE.</p> |
| RETURNS | M_BLK_ID or NULL . |
| ERRNO | S_netBufLib_MBLK_INVALID S_netBufLib_CLSIZE_INVALID S_netBufLib_NETPOOL_INVALID |
| SEE ALSO | netBufLib |

nfsAuthUnixGet()

- NAME** *nfsAuthUnixGet()* – get the NFS UNIX authentication parameters
- SYNOPSIS**
- ```
void nfsAuthUnixGet
(
 char * machname, /* where to store host machine */
 int * pUid, /* where to store user ID */
 int * pGid, /* where to store group ID */
 int * pNgids, /* where to store number of group IDs */
 int * gids /* where to store array of group IDs */
)
```
- DESCRIPTION** This routine gets the previously set UNIX authentication values.
- RETURNS** N/A
- SEE ALSO** *nfsLib, nfsAuthUnixPrompt(), nfsAuthUnixShow(), nfsAuthUnixSet(), nfsIdSet()*
- 

## ***nfsAuthUnixPrompt()***

- NAME** *nfsAuthUnixPrompt()* – modify the NFS UNIX authentication parameters
- SYNOPSIS**
- ```
void nfsAuthUnixPrompt (void)
```
- DESCRIPTION** This routine allows UNIX authentication parameters to be changed from the shell. The user is prompted for each parameter, which can be changed by entering the new value next to the current one.
- EXAMPLE**
- ```
-> nfsAuthUnixPrompt
machine name: yuba
user ID: 2001 128
group ID: 100
num of groups: 1 3
group #1: 100 100
group #2: 0 120
group #3: 0 200
value = 3 = 0x3
```
- SEE ALSO** *nfsLib, nfsAuthUnixShow(), nfsAuthUnixSet(), nfsAuthUnixGet(), nfsIdSet()*

---

## *nfsAuthUnixSet()*

|                    |                                                                                                                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>nfsAuthUnixSet()</i> – set the NFS UNIX authentication parameters                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>void nfsAuthUnixSet (     char * machname, /* host machine */     int   uid,       /* user ID */     int   gid,       /* group ID */     int   ngids,     /* number of group IDs */     int *  aup_gids  /* array of group IDs */ )</pre>                                   |
| <b>DESCRIPTION</b> | This routine sets UNIX authentication parameters. It is initially called by <i>usrNetInit()</i> in <i>usrConfig.c</i> . <b><i>machname</i></b> should be set with the name of the mounted system (i.e. the target name itself) to distinguish hosts from hosts on a NFS network. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <i>nfsLib</i> , <i>nfsAuthUnixPrompt()</i> , <i>nfsAuthUnixShow()</i> , <i>nfsAuthUnixGet()</i> , <i>nfsIdSet()</i> , <i>usrConfig</i>                                                                                                                                           |

---

## *nfsAuthUnixShow()*

|                    |                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>nfsAuthUnixShow()</i> – display the NFS UNIX authentication parameters                                                      |
| <b>SYNOPSIS</b>    | <pre>void nfsAuthUnixShow (void)</pre>                                                                                         |
| <b>DESCRIPTION</b> | This routine displays the parameters set by <i>nfsAuthUnixSet()</i> or <i>nfsAuthUnixPrompt()</i> .                            |
| <b>EXAMPLE</b>     | <pre>-&gt; nfsAuthUnixShow machine name = yuba user ID      = 2001 group ID     = 100 group [0]    = 100 value = 1 = 0x1</pre> |
| <b>RETURNS</b>     | N/A                                                                                                                            |
| <b>SEE ALSO</b>    | <i>nfsLib</i> , <i>nfsAuthUnixPrompt()</i> , <i>nfsAuthUnixSet()</i> , <i>nfsAuthUnixGet()</i> , <i>nfsIdSet()</i>             |

---

## *nfsDevInfoGet()*

**NAME** *nfsDevInfoGet()* – read configuration information from the requested NFS device

**SYNOPSIS**

```
STATUS nfsDevInfoGet
(
 unsigned long nfsDevHandle, /* NFS device handle */
 NFS_DEV_INFO * pnfsInfo /* ptr to struct to hold config info */
)
```

**DESCRIPTION** This routine accesses the NFS device specified in the parameter *nfsDevHandle* and fills in the structure pointed to by *pnfsInfo*.

**RETURNS** OK if *pnfsInfo* information is valid, otherwise ERROR.

**SEE ALSO** *nfsDrv*, *nfsDevListGet()*

---

## *nfsDevListGet()*

**NAME** *nfsDevListGet()* – create list of all the NFS devices in the system

**SYNOPSIS**

```
int nfsDevListGet
(
 unsigned long nfsDevList[], /* NFS dev list of handles */
 int listSize /* no. of elements available in the list */
)
```

**DESCRIPTION** This routine fills the array *nfsDevList* up to *listSize*, with handles to NFS devices currently in the system.

**RETURNS** The number of entries filled in the *nfsDevList* array.

**SEE ALSO** *nfsDrv*, *nfsDevInfoGet()*



---

## *nfsDevShow()*

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>nfsDevShow()</i> – display the mounted NFS devices                                                                              |
| <b>SYNOPSIS</b>    | <code>void nfsDevShow (void)</code>                                                                                                |
| <b>DESCRIPTION</b> | This routine displays the device names and their associated NFS file systems.                                                      |
| <b>EXAMPLE</b>     | <pre>-&gt; nfsDevShow device name          file system ----- /yuba1/              yuba:/yuba1 /wrs1/               wrs:/wrs1</pre> |
| <b>RETURNS</b>     | N/A                                                                                                                                |
| <b>SEE ALSO</b>    | <i>nfsDrv</i>                                                                                                                      |

---

## *nfsdInit()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>nfsdInit()</i> – initialize the NFS server                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS nfsdInit (     int    nServers,      /* the number of NFS servers to create */     int    nExportedFs,  /* maximum number of exported file systems */     int    priority,     /* the priority for the NFS servers */     FUNCPTR authHook,    /* authentication hook */     FUNCPTR mountAuthHook, /* authentication hook for mount daemon */     int    options       /* currently unused */ )</pre>                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | <p>This routine initializes the NFS server. <i>nServers</i> specifies the number of tasks to be spawned to handle NFS requests. <i>priority</i> is the priority that those tasks will run at. <i>authHook</i> is a pointer to an authorization routine. <i>mountAuthHook</i> is a pointer to a similar routine, passed to <i>mountdInit()</i>. <i>options</i> is provided for future expansion.</p> <p>Normally, no authorization is performed by either <i>mountd</i> or <i>nfsd</i>. If you want to add authorization, set <i>authHook</i> to a function pointer to a routine declared as follows:</p> |

***nfsDrv()***

*nfsstat* routine

```
(
 int progNum, /* RPC program number */
 int versNum, /* RPC program version number */
 int procNum, /* RPC procedure number */
 struct sockaddr_in clientAddr, /* address of the client */
 NFSD_ARGUMENT * nfsdArg /* argument of the call */
)
```

The *authHook* routine should return NFS\_OK if the request is authorized, and NFSERR\_ACCES if not. (NFSERR\_ACCES is not required; any legitimate NFS error code can be returned.)

See *mountdInit()* for documentation on *mountAuthHook*. Note that *mountAuthHook* and *authHook* can point to the same routine. Simply use the *progNum*, *versNum*, and *procNum* fields to decide whether the request is an NFS request or a mountd request.

**RETURNS** OK, or ERROR if the NFS server cannot be started.

**SEE ALSO** *nfsdLib*, *nfsExport()*, *mountdInit()*

***nfsDrv()***

**NAME** *nfsDrv()* – install the NFS driver

**SYNOPSIS** STATUS *nfsDrv* (void)

**DESCRIPTION** This routine initializes and installs the NFS driver. It must be called before any reads, writes, or other NFS calls. This is done automatically when the configuration macro INCLUDE\_NFS is defined.

**RETURNS** OK, or ERROR if there is no room for the driver.

**SEE ALSO** *nfsDrv*

***nfsDrvNumGet()***

**NAME** *nfsDrvNumGet()* – return the IO system driver number for the nfs driver

**SYNOPSIS** int *nfsDrvNumGet* (void)

**DESCRIPTION** This routine returns the nfs driver number allocated by iosDrvInstall during the nfs driver initialization. If the nfs driver has yet to be initialized, or if initialization failed, nfsDrvNumGet will return ERROR.

**RETURNS** the nfs driver number or ERROR

**SEE ALSO** nfsDrv

---

## *nfsdStatusGet()*

**NAME** *nfsdStatusGet()* – get the status of the NFS server

**SYNOPSIS**

```
STATUS nfsdStatusGet
(
 NFS_SERVER_STATUS * serverStats /* pointer to status structure */
)
```

**DESCRIPTION** This routine gets status information about the NFS server.

**RETURNS** OK, or ERROR if the information cannot be obtained.

**SEE ALSO** nfsdLib

---

## *nfsdStatusShow()*

**NAME** *nfsdStatusShow()* – show the status of the NFS server

**SYNOPSIS**

```
STATUS nfsdStatusShow
(
 int options /* unused */
)
```

**DESCRIPTION** This routine shows status information about the NFS server.

**RETURNS** OK, or ERROR if the information cannot be obtained.

**SEE ALSO** nfsdLib

## *nfsExport()*

**NAME** *nfsExport()* – specify a file system to be NFS exported

**SYNOPSIS**

```
STATUS nfsExport
(
 char * directory, /* Directory to export - FS must support NFS */
 int id, /* ID number for file system */
 BOOL readOnly, /* TRUE if file system is exported read-only */
 int options /* Reserved for future use - set to 0 */
)
```

**DESCRIPTION** This routine makes a file system available for mounting by a client. The client should be in the local host table (see *hostAdd()*), although this is not required.

The *id* parameter can either be set to a specific value, or to 0. If it is set to 0, an ID number is assigned sequentially. Every time a file system is exported, it must have the same ID number, or clients currently mounting the file system will not be able to access files.

To display a list of exported file systems, use:

```
-> nfsExportShow "localhost"
```

**RETURNS** OK, or ERROR if the file system could not be exported.

**SEE ALSO** *mountLib*, *nfsLib*, *nfsExportShow()*, *nfsUnexport()*

---

## *nfsExportShow()*

**NAME** *nfsExportShow()* – display the exported file systems of a remote host

**SYNOPSIS**

```
STATUS nfsExportShow
(
 char * hostName /* host machine to show exports for */
)
```

**DESCRIPTION** This routine displays the file systems of a specified host and the groups that are allowed to mount them.

**EXAMPLE**

```
-> nfsExportShow "wrs"
/d0 staff
/d1 staff eng
```

```

/d2 eng
/d3
value = 0 = 0x0

```

**RETURNS** OK or ERROR.

**SEE ALSO** `nfsLib`

---

## *nfsHelp()*

**NAME** `nfsHelp()` – display the NFS help menu

**SYNOPSIS** `void nfsHelp (void)`

**DESCRIPTION** This routine displays a summary of NFS facilities typically called from the shell:

|                                                          |                                                                              |
|----------------------------------------------------------|------------------------------------------------------------------------------|
| <code>nfsHelp</code>                                     | Print this list                                                              |
| <code>netHelp</code>                                     | Print general network help list                                              |
| <code>nfsMount "host", "filesystem" [, "devname"]</code> | Create device with<br>file system/directory from host                        |
| <code>nfsUnmount "devname"</code>                        | Remove an NFS device                                                         |
| <code>nfsAuthUnixShow</code>                             | Print current UNIX authentication                                            |
| <code>nfsAuthUnixPrompt</code>                           | Prompt for UNIX authentication                                               |
| <code>nfsIdSet id</code>                                 | Set user ID for UNIX authentication                                          |
| <code>nfsDevShow</code>                                  | Print list of NFS devices                                                    |
| <code>nfsExportShow "host"</code>                        | Print a list of NFS file systems which<br>are exported on the specified host |
| <code>mkdir "dirname"</code>                             | Create directory                                                             |
| <code>rm "file"</code>                                   | Remove file                                                                  |

EXAMPLE: `-> hostAdd "wrs", "90.0.0.2"`  
`-> nfsMount "wrs", "/disk0/path/mydir", "/mydir/"`  
`-> cd "/mydir/"`  
`-> nfsAuthUnixPrompt /* fill in user ID, etc. */`  
`-> ls /* list /disk0/path/mydir */`  
`-> copy < foo /* copy foo to standard out */`  
`-> ld < foo.o /* load object module foo.o */`  
`-> nfsUnmount "/mydir/" /* remove NFS device /mydir/ */`

**RETURNS** N/A

**SEE ALSO** `nfsLib`

***nfsIdSet()***

---

***nfsIdSet()***

|                    |                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>nfsIdSet()</i> – set the ID number of the NFS UNIX authentication parameters                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>void nfsIdSet (     int uid /* user ID on host machine */ )</pre>                                                                                                               |
| <b>DESCRIPTION</b> | This routine sets only the UNIX authentication user ID number. For most NFS permission needs, only the user ID needs to be changed. Set <i>uid</i> to the user ID on the NFS server. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <i>nfsLib</i> , <i>nfsAuthUnixPrompt()</i> , <i>nfsAuthUnixShow()</i> , <i>nfsAuthUnixSet()</i> , <i>nfsAuthUnixGet()</i>                                                            |

---

***nfsMount()***

|                    |                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>nfsMount()</i> – mount an NFS file system                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>STATUS nfsMount (     char * host,          /* name of remote host */     char * fileSystem,  /* name of remote directory to mount */     char * localName    /* local device name for remote dir (NULL = use */                         /* name) */ )</pre>                                         |
| <b>DESCRIPTION</b> | This routine mounts a remote file system. It creates a local device <i>localName</i> for a remote file system on a specified host. The host must have already been added to the local host table with <i>hostAdd()</i> . If <i>localName</i> is NULL, the local name will be the same as the remote name. |
| <b>RETURNS</b>     | OK, or ERROR if the driver is not installed, <i>host</i> is invalid, or memory is insufficient.                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>nfsDrv</i> , <i>nfsUnmount()</i> , <i>hostAdd()</i>                                                                                                                                                                                                                                                    |

---

## *nfsMountAll()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>nfsMountAll()</i> – mount all file systems exported by a specified host                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>STATUS nfsMountAll (     char * host,          /* name of remote host */     char * clientName, /* name of client specified in access list */     BOOL  quiet          /* FALSE = print names of file systems mounted */ )</pre>                                                                                                                                                                    |
| <b>DESCRIPTION</b> | <p>This routine mounts the file systems exported by <i>host</i> which are marked as accessible either by all clients or only by <i>clientName</i>. The <i>nfsMount()</i> routine is called to mount each file system. This creates a local device for each mounted file system that has the same name as the file system.</p> <p>The file systems are listed to standard output as they are mounted.</p> |
| <b>RETURNS</b>     | OK, or ERROR if any mount fails.                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <i>nfsDrv</i> , <i>nfsMount()</i>                                                                                                                                                                                                                                                                                                                                                                        |

---

## *nfsUnexport()*

|                    |                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>nfsUnexport()</i> – remove a file system from the list of exported file systems                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>STATUS nfsUnexport (     char * dirName /* Name of the directory to unexport */ )</pre>                                                                                                               |
| <b>DESCRIPTION</b> | <p>This routine removes a file system from the list of file systems exported from the target. Any client attempting to mount a file system that is not exported will receive an error (NFSERR_ACCESS).</p> |
| <b>RETURNS</b>     | OK, or ERROR if the file system could not be removed from the exports list.                                                                                                                                |
| <b>ERRNO</b>       | ENOENT                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <i>mountLib</i> , <i>nfsLib</i> , <i>nfsExportShow()</i> , <i>nfsExport()</i>                                                                                                                              |

## ***nfsUnmount()***

- NAME** *nfsUnmount()* – unmount an NFS device
- SYNOPSIS**
- ```
STATUS nfsUnmount
(
    char * localName /* local of nfs device */
)
```
- DESCRIPTION** This routine unmounts file systems that were previously mounted via NFS.
- RETURNS** OK, or ERROR if *localName* is not an NFS device or cannot be mounted.
- SEE ALSO** *nfsDrv*, *nfsMount()*
-

nicEndLoad()

- NAME** *nicEndLoad()* – initialize the driver and device
- SYNOPSIS**
- ```
END_OBJ* nicEvbEndLoad
(
 char* initString /* string to be parse by the driver */
)
```
- DESCRIPTION** This routine initializes the driver and the device to the operational state. All of the device-specific parameters are passed in *initString*, which expects a string of the following format:
- ```
unit:base_addr:int_vector:int_level
```
- This routine can be called in two modes. If it is called with an empty but allocated string, it places the name of this device (that is, "ln") into the *initString* and returns 0.
- If the string is allocated and not empty, the routine attempts to load the driver using the values specified in the string.
- RETURNS** An END object pointer, or NULL on error, or 0 and the name of the device if the *initString* was NULL.
- SEE ALSO** *nicEvbEnd*

nicEvbattach()

NAME *nicEvbattach()* – publish and initialize the **nicEvb** network interface driver

SYNOPSIS

```
STATUS nicEvbattach
(
    int          unit, /* unit number */
    NIC_DEVICE * pNic, /* address of NIC chip */
    int          ivec /* interrupt vector to use */
)
```

DESCRIPTION This routine publishes the **nicEvb** interface by filling in a network interface record and adding this record to the system list. It also initializes the driver and the device to the operational state.

RETURNS OK, or ERROR if the receive buffer memory could not be allocated.

SEE ALSO *if_nicEvb*

nicEvbInitParse()

NAME *nicEvbInitParse()* – parse the initialization string

SYNOPSIS

```
STATUS nicEvbInitParse
(
    NICEVB_END_DEVICE * pDrvCtrl,
    char *             initString
)
```

DESCRIPTION Parse the input string. Fill in values in the driver control structure. The initialization string format is: *unit:base_adrs:vecnum:intLvl*

unit
Device unit number, a small integer.

base_adrs
Base address for NIC device

vecNum
Interrupt vector number (used with *sysIntConnect()*).

nicTxStartup()

intLvl

Interrupt level.

RETURNS OK, or ERROR if any arguments are invalid.

SEE ALSO **nicEvbEnd**

nicTxStartup()

NAME *nicTxStartup()* – the driver’s actual output routine

SYNOPSIS

```
#ifdef BSD43_DRIVER LOCAL STATUS nicTxStartup
(
    int unit
)
```

DESCRIPTION This routine accepts outgoing packets from the `if_snd` queue, and then gains exclusive access to the DMA (through a mutex semaphore), then calls *nicTransmit()* to send the packet out onto the interface.

RETURNS OK, or ERROR if the packet could not be transmitted.

SEE ALSO **if_nicEvb**

npc()

NAME *npc()* – return the contents of the next program counter (SPARC)

SYNOPSIS

```
int npc
(
    int taskId /* task ID, 0 means default task */
)
```

DESCRIPTION This command extracts the contents of the next program counter from the TCB of a specified task. If *taskId* is omitted or 0, the current default task is assumed.

RETURNS The contents of the next program counter.

SEE ALSO **dbgArchLib**, *ti()*

ns16550DevInit()

NAME *ns16550DevInit()* – initialize an NS16550 channel

SYNOPSIS

```
void ns16550DevInit
(
    NS16550_CHAN * pChan /* pointer to channel */
)
```

DESCRIPTION This routine initializes some SIO_CHAN function pointers and then resets the chip in a quiescent state. Before this routine is called, the BSP must already have initialized all the device addresses, etc. in the NS16550_CHAN structure.

RETURNS N/A

SEE ALSO *ns16550Sio*

ns16550Int()

NAME *ns16550Int()* – interrupt level processing

SYNOPSIS

```
void ns16550Int
(
    NS16550_CHAN * pChan /* pointer to channel */
)
```

DESCRIPTION This routine handles four sources of interrupts from the UART. They are prioritized in the following order by the Interrupt Identification Register: Receiver Line Status, Received Data Ready, Transmit Holding Register Empty and Modem Status.

When a modem status interrupt occurs, the transmit interrupt is enabled if the CTS signal is TRUE.

RETURNS N/A

SEE ALSO *ns16550Sio*

ns16550IntEx()

NAME *ns16550IntEx()* – miscellaneous interrupt processing

SYNOPSIS

```
void ns16550IntEx  
(  
    NS16550_CHAN * pChan /* pointer to channel */  
)
```

DESCRIPTION This routine handles miscellaneous interrupts on the UART. Not implemented yet.

RETURNS N/A

SEE ALSO **ns16550Sio**

ns16550IntRd()

NAME *ns16550IntRd()* – handle a receiver interrupt

SYNOPSIS

```
void ns16550IntRd  
(  
    NS16550_CHAN * pChan /* pointer to channel */  
)
```

DESCRIPTION This routine handles read interrupts from the UART.

RETURNS N/A

SEE ALSO **ns16550Sio**

ns16550IntWr()

| | |
|--------------------|---|
| NAME | <i>ns16550IntWr()</i> – handle a transmitter interrupt |
| SYNOPSIS | <pre>void ns16550IntWr (NS16550_CHAN * pChan /* pointer to channel */)</pre> |
| DESCRIPTION | <p>This routine handles write interrupts from the UART. It reads a character and puts it in the transmit holding register of the device for transfer.</p> <p>If there are no more characters to transmit, transmission is disabled by clearing the transmit interrupt enable bit in the IER(int enable register).</p> |
| RETURNS | N/A |
| SEE ALSO | <i>ns16550Sio</i> |

ntInt()

| | |
|--------------------|--|
| NAME | <i>ntInt()</i> – handle controller interrupt |
| SYNOPSIS | <pre>void ntInt (NTEND_DEVICE * pDrvCtrl)</pre> |
| DESCRIPTION | This routine is called at interrupt level in response to an interrupt from the controller. |
| RETURNS | N/A. |
| SEE ALSO | <i>ntEnd</i> |

ntLoad()

ntLoad()

NAME *ntLoad()* – initialize the driver and device

SYNOPSIS

```
END_OBJ* ntLoad
(
    char* initString, /* String to be parse by the driver. */
    void* nothing
)
```

DESCRIPTION This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the *initString*.

The string contains the target specific parameters like this:

"unit:register addr:int vector:int level:shmem addr:shmem size:shmem width"

RETURNS An END object pointer or NULL on error.

SEE ALSO *ntEnd*

ntMemInit()

NAME *ntMemInit()* – initialize memory for the chip

SYNOPSIS

```
STATUS ntMemInit
(
    NTEND_DEVICE * pDrvCtrl /* device to be initialized */
)
```

DESCRIPTION This routine is highly specific to the device.

RETURNS OK or ERROR.

SEE ALSO *ntEnd*

ntParse()

| | |
|--------------------|---|
| NAME | <i>ntParse()</i> – parse the init string |
| SYNOPSIS | <pre>STATUS ntParse (NTEND_DEVICE * pDrvCtrl, char * initString)</pre> |
| DESCRIPTION | <p>Parse the input string. Fill in values in the driver control structure. The initialization string format is:</p> <pre>"unit:csrAdr:rapAdr:vecnum:intLvl:memAdrs:memSize:memWidth"</pre> <p><i>unit</i> Device unit number, a small integer.</p> <p><i>vecNum</i> Interrupt vector number (used with <i>sysIntConnect()</i>)</p> <p><i>intLvl</i> Interrupt level (isn't really used)</p> |
| RETURNS | OK or ERROR for invalid arguments. |
| SEE ALSO | <i>ntEnd</i> |

ntPassFsDevInit()

| | |
|--------------------|---|
| NAME | <i>ntPassFsDevInit()</i> – associate a device with ntPassFs file system functions |
| SYNOPSIS | <pre>void *ntPassFsDevInit (char * devName /* device name */)</pre> |
| DESCRIPTION | <p>This routine associates the name <i>devName</i> with the file system and installs it in the I/O System's device table. The driver number used when the device is added to the table is that which was assigned to the ntPassFs library during <i>ntPassFsInit()</i>.</p> |
| RETURNS | A pointer to the volume descriptor, or NULL if there is an error. |
| SEE ALSO | <i>ntPassFsLib</i> |

ntPassFsInit()

| | |
|--------------------|--|
| NAME | <i>ntPassFsInit()</i> – prepare to use the ntPassFs library |
| SYNOPSIS | <pre>STATUS ntPassFsInit (int nPassfs /* number of ntPass-through file systems */)</pre> |
| DESCRIPTION | <p>This routine initializes the ntPassFs library. It must be called exactly once, before any other routines in the library. The argument specifies the number of ntPassFs devices that may be open at once. This routine installs ntPassFsLib as a driver in the I/O system driver table, allocates and sets up the necessary memory structures, and initializes semaphores.</p> <p>Normally this routine is called from the root task, <i>usrRoot()</i>, in <i>usrConfig()</i>. To enable this initialization, define INCLUDE_PASSFS in configAll.h.</p> |
| NOTE | Maximum number of ntPass-through file systems is 1. |
| RETURNS | OK, or ERROR. |
| SEE ALSO | ntPassFsLib |

ntPollStart()

| | |
|-----------------|--|
| NAME | <i>ntPollStart()</i> – start polled mode operations |
| SYNOPSIS | <pre>STATUS ntPollstart (NTEND_DEVICE* pDrvCtrl)</pre> |
| RETURNS | OK or ERROR. |
| SEE ALSO | ntEnd |

ntPollStop()

| | |
|--------------------|---|
| NAME | <i>ntPollStop()</i> – stop polled mode operations |
| SYNOPSIS | <pre> STATUS ntPollStop (NTEND_DEVICE* pDrvCtrl) </pre> |
| DESCRIPTION | <p>This function terminates polled mode operation. The device returns to interrupt mode. The device interrupts are enabled, the current mode flag is switched to indicate interrupt mode and the device is then reconfigured for interrupt operation.</p> |
| RETURNS | OK or ERROR. |
| SEE ALSO | <i>ntEnd</i> |

o0()

| | |
|--------------------|---|
| NAME | <i>o0()</i> – return the contents of register o0 (also o1 – o7) (SPARC) |
| SYNOPSIS | <pre> int o0 (int taskId /* task ID, 0 means default task */) </pre> |
| DESCRIPTION | <p>This command extracts the contents of out register o0 from the TCB of a specified task. If <i>taskId</i> is omitted or 0, the current default task is assumed.</p> <p>Similar routines are provided for all out registers (o0 – o7): <i>o0()</i> – <i>o7()</i>.</p> <p>The stack pointer is accessed via o6.</p> |
| RETURNS | The contents of register o0 (or the requested register). |
| SEE ALSO | dbgArchLib , <i>VxWorks Programmer's Guide: Target Shell</i> |

open()

open()**NAME** *open()* – open a file

SYNOPSIS

```
int open
(
    const char * name, /* name of the file to open */
    int         flags, /* O_RDONLY, O_WRONLY, O_RDWR, or O_CREAT */
    int         mode  /* mode of file to create (UNIX chmod style) */
)
```

DESCRIPTION This routine opens a file for reading, writing, or updating, and returns a file descriptor for that file. The arguments to *open()* are the filename and the type of access:

O_RDONLY (0) (or READ) - open for reading only.
 O_WRONLY (1) (or WRITE) - open for writing only.
 O_RDWR (2) (or UPDATE) - open for reading and writing.
 O_CREAT (0x0200) - create a file.

In general, *open()* can only open pre-existing devices and files. However, for NFS network devices only, files can also be created with *open()* by performing a logical OR operation with O_CREAT and the *flags* argument. In this case, the file is created with a UNIX chmod-style file mode, as indicated with *mode*. For example:

```
fd = open ("/usr/myFile", O_CREAT | O_RDWR, 0644);
```

Only the NFS driver uses the *mode* argument.

NOTE For more information about situations when there are no file descriptors available, see the manual entry for *iosInit()*.

RETURNS A file descriptor number, or ERROR if a file name is not specified, the device does not exist, no file descriptors are available, or the driver returns ERROR.

ERRNO ELOOP

SEE ALSO *ioLib*, *creat()*

opendir()

| | |
|--------------------|--|
| NAME | <i>opendir()</i> – open a directory for searching (POSIX) |
| SYNOPSIS | <pre>DIR *opendir (char * dirName /* name of directory to open */)</pre> |
| DESCRIPTION | <p>This routine opens the directory named by <i>dirName</i> and allocates a directory descriptor (DIR) for it. A pointer to the DIR structure is returned. The return of a NULL pointer indicates an error.</p> <p>After the directory is opened, <i>readdir()</i> is used to extract individual directory entries. Finally, <i>closedir()</i> is used to close the directory.</p> |
| WARNING | For remote file systems mounted over netDrv , <i>opendir()</i> fails, because the netDrv implementation strategy does not provide a way to distinguish directories from plain files. To permit use of <i>opendir()</i> on remote files, use NFS rather than netDrv. |
| RETURNS | A pointer to a directory descriptor, or NULL if there is an error. |
| SEE ALSO | <i>dirLib</i> , <i>closedir()</i> , <i>readdir()</i> , <i>rewinddir()</i> , <i>ls()</i> |

operator delete()

| | |
|--------------------|--|
| NAME | <i>operator delete()</i> – default run-time support for memory deallocation (C++) |
| SYNOPSIS | <pre>extern void operator delete (void * pMem /* pointer to dynamically-allocated object */)</pre> |
| DESCRIPTION | This function provides the default implementation of operator delete. It returns the memory, previously allocated by operator new, to the VxWorks system memory partition. |
| RETURNS | N/A |
| SEE ALSO | <i>cplusplusLib</i> |

operator new()

NAME *operator new()* – default run-time support for operator new (C++)

SYNOPSIS

```
extern void * operator new
(
    size_t n /* size of object to allocate */
) throw (std::bad_alloc)
```

DESCRIPTION This function provides the default implementation of operator new. It allocates memory from the system memory partition for the requested object. The value, when evaluated, is a pointer of the type **pointer-to-*T*** where *T* is the type of the new object.

If allocation fails a new-handler, if one is defined, is called. If the new-handler returns, presumably after attempting to recover from the memory allocation failure, allocation is retried. If there is no new-handler an exception of type "bad_alloc" is thrown.

RETURNS Pointer to new object.

THROWS std::bad_alloc if allocation failed.

SEE ALSO cplusplus

operator new()

NAME *operator new()* – default run-time support for operator new (nothrow) (C++)

SYNOPSIS

```
extern void * operator new
(
    size_t          n, /* size of object to allocate */
    const nothrow_t & /* supply argument of "nothrow" here */
) throw ()
```

DESCRIPTION This function provides the default implementation of operator new (nothrow). It allocates memory from the system memory partition for the requested object. The value, when evaluated, is a pointer of the type **pointer-to-*T*** where *T* is the type of the new object.

If allocation fails, a new-handler, if one is defined, is called. If the new-handler returns, presumably after attempting to recover from the memory allocation failure, allocation is retried. If the new_handler throws a bad_alloc exception, the exception is caught and 0 is returned. If allocation fails and there is no new_handler 0 is returned.

RETURNS Pointer to new object or 0 if allocation fails.

INCLUDE FILES `new`

SEE ALSO `cplusplus`

operator new()

NAME *operator new()* – run-time support for operator new with placement (C++)

SYNOPSIS

```
extern void * operator new
(
    size_t n, /* size of object to allocate (unused) */
    void * pMem /* pointer to allocated memory */
)
```

DESCRIPTION This function provides the default implementation of the global new operator, with support for the placement syntax. New-with-placement is used to initialize objects for which memory has already been allocated. *pMem* points to the previously allocated memory.

RETURNS *pMem*

INCLUDE FILES `new`

SEE ALSO `cplusplus`

ospfExtRouteAdd()

NAME *ospfExtRouteAdd()* – import external route into OSPF domain (OSPF Opt.)

SYNOPSIS

```
STATUS ospfExtRouteAdd
(
    uint32_t destIp, /* destination IP address */
    uint32_t destMask, /* destination mask */
    uint32_t nextHopIp, /* IP address of next hop */
    int cost, /* cost to advertise in domain */
    int extRouteType, /* 1 = external type1, 2 = external type2 */
    int tos /* type of service */
)
```

DESCRIPTION This function is used to import an external route into the OSPF domain. The destination address and mask are *destIp* and *destMask* respectively while *nextHopIp* is the IP address of the next hop. The cost to advertise in the OSPF domain is *cost* and route type is *routeType*, which can have the value 1 or 2 for type 1 and type 2 routes respectively. All IP addresses and masks in this call are in network byte order.

RETURNS OK or ERROR.

SEE ALSO **ospfLib**

ospfExtRouteDelete()

NAME *ospfExtRouteDelete()* – delete external route imported into OSPF (OSPF Opt.)

SYNOPSIS

```
STATUS ospfExtRouteDelete
(
    uint32_t destIp,          /* destination IP address */
    uint32_t destMask,       /* destination mask */
    int      extRouteType,   /* 1 = external type1, 2 = external type2 */
    int      tos              /* type of service */
)
```

DESCRIPTION This function is used to delete an external route imported into the OSPF domain. The destination address and mask are *destIp* and *destMask* respectively. The route type is *extRouteType* which may have the value 1 or 2 for type 1 and type 2 routes, respectively. All IP addresses and masks in this call are in network byte order.

RETURNS OK or ERROR.

SEE ALSO **ospfLib**

ospfInit()

NAME *ospfInit()* – function to initialize OSPF routing (OSPF Opt.)

SYNOPSIS

```
STATUS ospfInit
(
    int      priority, /* task priority */
    int      options,  /* task options */
    int      stackSize, /* task stack size */
)
```

```

int    routerId, /* routerId, host byte order */
FUNCPTR pAuthHook /* ospf authentication hook */
)

```

DESCRIPTION

This function initializes the OSPF facilities. This includes creating OSPF tasks, which are created with a priority of *priority*, options set to *options*, a stack size of *stackSize*, and an OSPF router ID of *routerid*. If *routerId* is 0, the IP address of one of the interfaces is used as the router ID. The *pAuthHook* parameter expects a pointer to a user-provided authentication routine. For every received packet, the authentication function:

```
(*ospfAuthHook) (pIfkey, pPktKey, ipAddr)
```

The *pIfkey* parameter is a pointer to the authorization key associated with the interface. The *pPktKey* parameter is a pointer to the key in the received packet. The *ipAddr* is the IP address in network byte order of the interface on which the packet was received. To set the interface authorization key, call *m2OspfIfEntrySet()*. The *ospfAuthHook()* routine returns TRUE if the packet is acceptable. Otherwise, it returns FALSE.

After this function has returned, you can use the *m2Ospf*Set()* configuration routines to alter the settings.

RETURNS

OK or ERROR.

SEE ALSO

ospfLib

ospfNbmaDstAdd()

NAME

ospfNbmaDstAdd() – add NBMA destination

SYNOPSIS

```

STATUS ospfNbmaDstAdd
(
    uint32_t ipAddress, /* neighbor IP address, network order */
    uint32_t ifIpAddress, /* local interface IP address, network order */
    BOOL    eligible /* TRUE if neighbor is eligible to be DR */
)

```

DESCRIPTION

On a non-broadcast multiple access network, a router capable of becoming designated router must be made aware of the IP addresses of all other routers on the network. The neighbor router is specified by its IP address *ipAddress*, the local interface IP address is *ifIpAddress* and *eligible* specifies if the neighbor is capable of acting as a designated router.

RETURNS

OK or ERROR.

SEE ALSO

ospfLib

ospfNbmaDstDelete()

| | |
|--------------------|--|
| NAME | <i>ospfNbmaDstDelete()</i> – delete NBMA destination |
| SYNOPSIS | <pre>STATUS ospfNbmaDstDelete (uint32_t ipAddress, /* neighbor IP address, network order */ uint32_t ifIpAddress /* local interface IP address, network order */)</pre> |
| DESCRIPTION | Delete neighbor on a NBMA network, previously created with <i>ospfNbmaDstAdd()</i> . The neighbor is specified by its IP address <i>ipAddress</i> and the local interface IP address is <i>ifIpAddress</i> . |
| RETURNS | OK or ERROR. |
| SEE ALSO | ospfLib |

ospfTerminate()

| | |
|--------------------|--|
| NAME | <i>ospfTerminate()</i> – free OSPF resources and delete OSPF tasks |
| SYNOPSIS | <pre>void ospfTerminate ()</pre> |
| DESCRIPTION | This function frees all the resources used by OSPF. This includes deleting the two VxWorks tasks used to manage OSPF. You are free to restart OSPF after this function has returned. |
| RETURNS | N/A |
| SEE ALSO | ospfLib |

passFsDevInit()

| | |
|--------------------|---|
| NAME | <i>passFsDevInit()</i> – associate a device with passFs file system functions |
| SYNOPSIS | <pre>void *passFsDevInit (char * devName /* device name */)</pre> |
| DESCRIPTION | This routine associates the name <i>devName</i> with the file system and installs it in the I/O System's device table. The driver number used when the device is added to the table is that which was assigned to the passFs library during <i>passFsInit()</i> . |
| RETURNS | A pointer to the volume descriptor, or NULL if there is an error. |
| SEE ALSO | passFsLib |

passFsInit()

| | |
|--------------------|---|
| NAME | <i>passFsInit()</i> – prepare to use the passFs library |
| SYNOPSIS | <pre>STATUS passFsInit (int nPassfs /* number of pass-through file systems */)</pre> |
| DESCRIPTION | <p>This routine initializes the passFs library. It must be called exactly once, before any other routines in the library. The argument specifies the number of passFs devices that may be open at once. This routine installs passFsLib as a driver in the I/O system driver table, allocates and sets up the necessary memory structures, and initializes semaphores.</p> <p>Normally this routine is called from the root task, <i>usrRoot()</i>, in <i>usrConfig()</i>. This initialization is enabled when the configuration macro INCLUDE_PASSFS is defined.</p> |
| NOTE | Maximum number of pass-through file systems is 1. |
| RETURNS | OK, or ERROR. |
| SEE ALSO | passFsLib |

pause()

pause()

NAME *pause()* – suspend the task until delivery of a signal (POSIX)

SYNOPSIS `int pause (void)`

DESCRIPTION This routine suspends the task until delivery of a signal.

NOTE Since the *pause()* function suspends thread execution indefinitely, there is no successful completion return value.

RETURNS -1, always.

ERRNO EINTR

SEE ALSO sigLib

pc()

NAME *pc()* – return the contents of the program counter

SYNOPSIS `int pc
(
 int task /* task ID */
)`

DESCRIPTION This command extracts the contents of the program counter for a specified task from the task's TCB. If *task* is omitted or 0, the current task is used.

RETURNS The contents of the program counter.

SEE ALSO usrLib, *ti()*, *VxWorks Programmer's Guide: Target Shell*

pccardAtaEnabler()

NAME *pccardAtaEnabler()* – enable the PCMCIA-ATA device

SYNOPSIS

```
STATUS pccardAtaEnabler
(
    int          sock,          /* socket no. */
    ATA_RESOURCE * pAtaResource, /* pointer to ATA resources */
    int          numEnt,       /* number of ATA resource entries */
    FUNCPTR      showRtn      /* ATA show routine */
)
```

DESCRIPTION This routine enables the PCMCIA-ATA device.

RETURNS OK, `ERROR_FIND` if there is no ATA card, or `ERROR` if another error occurs.

SEE ALSO `pccardLib`

pccardEltEnabler()

NAME *pccardEltEnabler()* – enable the PCMCIA Etherlink III card

SYNOPSIS

```
STATUS pccardEltEnabler
(
    int          sock,          /* socket no. */
    ELT_RESOURCE * pEltResource, /* pointer to ELT resources */
    int          numEnt,       /* number of ELT resource entries */
    FUNCPTR      showRtn      /* show routine */
)
```

DESCRIPTION This routine enables the PCMCIA Etherlink III (ELT) card.

RETURNS OK, `ERROR_FIND` if there is no ELT card, or `ERROR` if another error occurs.

SEE ALSO `pccardLib`

pccardMkfs()

pccardMkfs()

NAME *pccardMkfs()* – initialize a device and mount a DOS file system

SYNOPSIS `STATUS pccardMkfs`
`(`
`int sock, /* socket number */`
`char * pName /* name of a device */`
`)`

DESCRIPTION This routine initializes a device and mounts a DOS file system.

RETURNS OK or ERROR.

SEE ALSO *pccardLib*

pccardMount()

NAME *pccardMount()* – mount a DOS file system

SYNOPSIS `STATUS pccardMount`
`(`
`int sock, /* socket number */`
`char * pName /* name of a device */`
`)`

DESCRIPTION This routine mounts a DOS file system.

RETURNS OK or ERROR.

SEE ALSO *pccardLib*

pccardSramEnabler()

NAME *pccardSramEnabler()* – enable the PCMCIA-SRAM driver

SYNOPSIS

```
STATUS pccardSramEnabler
(
    int          sock,          /* socket no. */
    SRAM_RESOURCE * pSramResource, /* pointer to SRAM resources */
    int          numEnt,        /* number of SRAM resource entries */
    FUNCPTR      showRtn        /* SRAM show routine */
)
```

DESCRIPTION This routine enables the PCMCIA-SRAM driver.

RETURNS OK, ERROR_FIND if there is no SRAM card, or ERROR if another error occurs.

SEE ALSO *pccardLib*

pccardTffsEnabler()

NAME *pccardTffsEnabler()* – enable the PCMCIA-TFFS driver

SYNOPSIS

```
STATUS pccardTffsEnabler
(
    int          sock,          /* socket no. */
    TFFS_RESOURCE * pTffsResource, /* pointer to TFFS resources */
    int          numEnt,        /* number of SRAM resource entries */
    FUNCPTR      showRtn        /* TFFS show routine */
)
```

DESCRIPTION This routine enables the PCMCIA-TFFS driver.

RETURNS OK, ERROR_FIND if there is no TFFS(Flash) card, or ERROR if another error occurs.

SEE ALSO *pccardLib*

pcicInit()

pcicInit()

NAME *pcicInit()* – initialize the PCIC chip

SYNOPSIS

```
STATUS pcicInit
(
    int    ioBase,    /* IO base address */
    int    intVec,    /* interrupt vector */
    int    intLevel, /* interrupt level */
    FUNCPTR showRtn  /* show routine */
)
```

DESCRIPTION This routine initializes the PCIC chip.

RETURNS OK, or ERROR if the PCIC chip cannot be found.

SEE ALSO *pcic*

pcicShow()

NAME *pcicShow()* – show all configurations of the PCIC chip

SYNOPSIS

```
void pcicShow
(
    int sock /* socket no. */
)
```

DESCRIPTION This routine shows all configurations of the PCIC chip.

RETURNS N/A

SEE ALSO *pcicShow*

pcmciaad()

| | |
|--------------------|---|
| NAME | <i>pcmciaad()</i> – handle task-level PCMCIA events |
| SYNOPSIS | <code>void pcmciaad (void)</code> |
| DESCRIPTION | This routine is spawned as a task by <i>pcmciaInit()</i> to perform functions that cannot be performed at interrupt or trap level. It has a priority of 0. Do not suspend, delete, or change the priority of this task. |
| RETURNS | N/A |
| SEE ALSO | <i>pcmciaLib</i> , <i>pcmciaInit()</i> |

pcmciaInit()

| | |
|--------------------|---|
| NAME | <i>pcmciaInit()</i> – initialize the PCMCIA event-handling package |
| SYNOPSIS | <code>STATUS pcmciaInit (void)</code> |
| DESCRIPTION | This routine installs the PCMCIA event-handling facilities and spawns <i>pcmciaad()</i> , which performs special PCMCIA event-handling functions that need to be done at task level. It also creates the message queue used to communicate with <i>pcmciaad()</i> . |
| RETURNS | OK, or ERROR if a message queue cannot be created or <i>pcmciaad()</i> cannot be spawned. |
| SEE ALSO | <i>pcmciaLib</i> , <i>pcmciaad()</i> |

pcmciaShow()

| | |
|-----------------|--|
| NAME | <i>pcmciaShow()</i> – show all configurations of the PCMCIA chip |
| SYNOPSIS | <pre>void pcmciaShow (int sock /* socket no. */)</pre> |

DESCRIPTION This routine shows all configurations of the PCMCIA chip.

RETURNS N/A

SEE ALSO **pcmciaShow**

pcmciaShowInit()

NAME *pcmciaShowInit()* – initialize all show routines for PCMCIA drivers

SYNOPSIS `void pcmciaShowInit (void)`

DESCRIPTION This routine initializes all show routines related to PCMCIA drivers.

RETURNS N/A

SEE ALSO **pcmciaShow**

pcw()

NAME *pcw()* – return the contents of the **pcw** register (i960)

SYNOPSIS `int pcw
(
 int taskId /* task ID, 0 means default task */
)`

DESCRIPTION This command extracts the contents of the **pcw** register from the TCB of a specified task. If *taskId* is omitted or 0, the current default task is assumed.

RETURNS The contents of the **pcw** register.

SEE ALSO **dbgArchLib**, *VxWorks Programmer's Guide: Target Shell*

pentiumBtc()

NAME *pentiumBtc()* – execute atomic compare-and-exchange instruction to clear a bit

SYNOPSIS

```
STATUS pentiumBtc
(
)
char * pFlag; /* flag address */
```

DESCRIPTION This routine compares a byte specified by the first parameter with TRUE. If it is TRUE, it changes it to 0 and returns OK. If it is not TRUE, it returns ERROR. LOCK and CMPXCHGB are used to get the atomic memory access.

RETURNS OK or ERROR if the specified flag is not TRUE

SEE ALSO *pentiumALib*

pentiumBts()

NAME *pentiumBts()* – execute atomic compare-and-exchange instruction to set a bit

SYNOPSIS

```
STATUS pentiumBts
(
)
char * pFlag; /* flag address */
```

DESCRIPTION This routine compares a byte specified by the first parameter with 0. If it is 0, it changes it to TRUE and returns OK. If it is not 0, it returns ERROR. LOCK and CMPXCHGB are used to get the atomic memory access.

RETURNS OK or ERROR if the specified flag is not zero

SEE ALSO *pentiumALib*

pentiumCr4Get()

NAME *pentiumCr4Get()* – Get a content of CR4 register

SYNOPSIS `int pentiumCr4Get (void)`

DESCRIPTION This routine gets a content of CR4 register.

RETURNS a content of CR4 register

SEE ALSO `pentiumALib`

pentiumCr4Set()

NAME *pentiumCr4Set()* – Set a specified value to CR4 register

SYNOPSIS `void pentiumCr4Set
(
)
int cr4; /* value to write CR4 register */`

DESCRIPTION This routine sets a specified value to CR4 register.

RETURNS N/A

SEE ALSO `pentiumALib`

pentiumMcaShow()

NAME *pentiumMcaShow()* – show MCA (Machine Check Architecture) registers

SYNOPSIS `void pentiumMcaShow (void)`

DESCRIPTION This routine shows Machine-Check global control registers and Error-Reporting register banks. Number of the Error-Reporting register banks is kept in a variable `mcaBanks`. `MCi_ADDR` and `MCi_MISC` registers in the Error-Reporting register bank are showed if `MCi_STATUS` indicates that these registers are valid.

RETURNS N/A
SEE ALSO `pentiumShow`

pentiumMsrGet()

NAME *pentiumMsrGet()* – get a content of the specified MSR (Model Specific Register)

SYNOPSIS

```
void pentiumMsrGet  
(  
)  
int addr; /* MSR address */  
long long int * pData; /* MSR data */
```

DESCRIPTION This routine gets a content of the specified MSR. The first parameter is an address of the MSR. The second parameter is a pointer of 64Bit variable.

RETURNS N/A

SEE ALSO `pentiumALib`

pentiumMsrSet()

NAME *pentiumMsrSet()* – set a value to the specified MSR (Model Specific Registers)

SYNOPSIS

```
void pentiumMsrSet  
(  
)  
int addr; /* MSR address */  
long long int * pData; /* MSR data */
```

DESCRIPTION This routine sets a value to a specified MSR. The first parameter is an address of the MSR. The second parameter is a pointer of 64Bit variable.

RETURNS N/A

SEE ALSO `pentiumALib`

pentiumMtrrDisable()

| | |
|--------------------|---|
| NAME | <i>pentiumMtrrDisable()</i> – disable MTRR (Memory Type Range Register) |
| SYNOPSIS | <code>void pentiumMtrrDisable (void)</code> |
| DESCRIPTION | This routine disables the MTRR that provide a mechanism for associating the memory types with physical address ranges in system memory. |
| RETURNS | N/A |
| SEE ALSO | <code>pentiumLib</code> |

pentiumMtrrEnable()

| | |
|--------------------|--|
| NAME | <i>pentiumMtrrEnable()</i> – enable MTRR (Memory Type Range Register) |
| SYNOPSIS | <code>void pentiumMtrrEnable (void)</code> |
| DESCRIPTION | This routine enables the MTRR that provide a mechanism for associating the memory types with physical address ranges in system memory. |
| RETURNS | N/A |
| SEE ALSO | <code>pentiumLib</code> |

pentiumMtrrGet()

| | |
|-----------------|--|
| NAME | <i>pentiumMtrrGet()</i> – get MTRRs to a specified MTRR table |
| SYNOPSIS | <pre>STATUS pentiumMtrrGet (MTRR * pMtrr /* MTRR table */)</pre> |

DESCRIPTION This routine gets MTRRs to a specified MTRR table with RDMSR instruction. The read MTRRs are CAP register, DEFTYPE register, fixed range MTRRs, and variable range MTRRs.

RETURNS OK, or ERROR if MTRR is being accessed.

SEE ALSO **pentiumLib**

pentiumMtrrSet()

NAME *pentiumMtrrSet()* – set MTRRs from specified MTRR table with WRMSR instruction.

SYNOPSIS

```
STATUS pentiumMtrrSet
(
    MTRR * pMtrr /* MTRR table */
)
```

DESCRIPTION This routine sets MTRRs from specified MTRR table with WRMSR instruction. The written MTRRs are DEFTYPE register, fixed range MTRRs, and variable range MTRRs.

RETURNS OK, or ERROR if MTRR is enabled or being accessed.

SEE ALSO **pentiumLib**

pentiumPmcGet()

NAME *pentiumPmcGet()* – get contents of PMC0 and PMC1

SYNOPSIS

```
void pentiumPmcGet
(
    long long int * pPmc0; /* Performance Monitoring Counter 0 */
    long long int * pPmc1; /* Performance Monitoring Counter 1 */
)
```

DESCRIPTION This routine gets contents of both PMC0 (Performance Monitoring Counter 0) and PMC1. The first parameter is a pointer of 64Bit variable to store the content of the Counter 0, and the second parameter is for the Counter 1.

RETURNS N/A

SEE ALSO **pentiumALib**

pentiumPmcGet0()

NAME *pentiumPmcGet0()* – get a content of PMC0

SYNOPSIS

```
void pentiumPmcGet0
(
)
long long int * pPmc0; /* Performance Monitoring Counter 0 */
```

DESCRIPTION This routine gets a content of PMC0 (Performance Monitoring Counter 0). Parameter is a pointer of 64Bit variable to store the content of the Counter.

RETURNS N/A

SEE ALSO **pentiumALib**

pentiumPmcGet1()

NAME *pentiumPmcGet1()* – get a content of PMC1

SYNOPSIS

```
void pentiumPmcGet1
(
)
long long int * pPmc1; /* Performance Monitoring Counter 1 */
```

DESCRIPTION This routine gets a content of PMC1 (Performance Monitoring Counter 1). Parameter is a pointer of 64Bit variable to store the content of the Counter.

RETURNS N/A

SEE ALSO **pentiumALib**

pentiumPmcReset()

NAME *pentiumPmcReset()* – reset both PMC0 and PMC1

SYNOPSIS `void pentiumPmcReset (void)`

DESCRIPTION This routine resets both PMC0 (Performance Monitoring Counter 0) and PMC1.

RETURNS N/A

SEE ALSO *pentiumALib*

pentiumPmcReset0()

NAME *pentiumPmcReset0()* – reset PMC0

SYNOPSIS `void pentiumPmcReset0 (void)`

DESCRIPTION This routine resets PMC0 (Performance Monitoring Counter 0).

RETURNS N/A

SEE ALSO *pentiumALib*

pentiumPmcReset1()

NAME *pentiumPmcReset1()* – reset PMC1

SYNOPSIS `void pentiumPmcReset1 (void)`

DESCRIPTION This routine resets PMC1 (Performance Monitoring Counter 1).

RETURNS N/A

SEE ALSO *pentiumALib*

pentiumPmcShow()

NAME *pentiumPmcShow()* – show PMCs (Performance Monitoring Counters)

SYNOPSIS

```
void pentiumPmcShow
(
    BOOL zap /* 1: reset PMC0 and PMC1 */
)
```

DESCRIPTION This routine shows Performance Monitoring Counter 0 and 1. Monitored events are selected by Performance Event Select Registers in *pentiumPmcStart()*. These counters are cleared to 0 if the parameter "zap" is TRUE.

RETURNS N/A

SEE ALSO *pentiumShow*

pentiumPmcStart()

NAME *pentiumPmcStart()* – start both PMC0 and PMC1

SYNOPSIS

```
STATUS pentiumPmcStart
(
    int pmcEvtSel0; /* Performance Event Select Register 0 */
    int pmcEvtSel1; /* Performance Event Select Register 1 */
)
```

DESCRIPTION This routine starts both PMC0 (Performance Monitoring Counter 0) and PMC1 by writing specified events to Performance Event Select Registers. The first parameter is a content of Performance Event Select Register 0, and the second parameter is for the Performance Event Select Register 1.

RETURNS OK or ERROR if PMC is already started

SEE ALSO *pentiumALib*

pentiumPmcStop()

| | |
|--------------------|--|
| NAME | <i>pentiumPmcStop()</i> – stop both PMC0 and PMC1 |
| SYNOPSIS | <code>void pentiumPmcStop (void)</code> |
| DESCRIPTION | This routine stops both PMC0 (Performance Monitoring Counter 0) and PMC1 by clearing two Performance Event Select Registers. |
| RETURNS | N/A |
| SEE ALSO | <i>pentiumALib</i> |

pentiumPmcStop1()

| | |
|--------------------|---|
| NAME | <i>pentiumPmcStop1()</i> – stop PMC1 |
| SYNOPSIS | <code>void pentiumPmcStop1 (void)</code> |
| DESCRIPTION | This routine stops only PMC1 (Performance Monitoring Counter 1) by clearing the Performance Event Select Register 1. Note, clearing the Performance Event Select Register 0 stops both counters, PMC0 and PMC1. |
| RETURNS | N/A |
| SEE ALSO | <i>pentiumALib</i> |

pentiumSerialize()

| | |
|--------------------|--|
| NAME | <i>pentiumSerialize()</i> – execute a serializing instruction CPUID |
| SYNOPSIS | <code>void pentiumSerialize (void)</code> |
| DESCRIPTION | This routine executes a serializing instruction CPUID. Serialization means that all modifications to flags, registers, and memory by previous instructions are completed before the next instruction is fetched and executed and all buffered writes have drained to memory. |

RETURNS N/A

SEE ALSO **pentiumALib**

pentiumTlbFlush()

NAME *pentiumTlbFlush()* – flush TLBs (Translation Lookaside Buffers)

SYNOPSIS `void pentiumTlbFlush (void)`

DESCRIPTION This routine flushes TLBs by loading CR3 register. All of the TLBs are automatically invalidated any time the CR3 register loaded. The page global enable (PGE) flag in register CR4 and the global flag in a page-directory or page-table entry can be used to frequently used pages from being automatically invalidated in the TLBs on a load of CR3 register. The only way to deterministically invalidate global page entries is to clear the PGE flag and then invalidate the TLBs.

RETURNS N/A

SEE ALSO **pentiumALib**

pentiumTscGet32()

NAME *pentiumTscGet32()* – get a lower half of the 64Bit TSC (Timestamp Counter)

SYNOPSIS `UINT32 pentiumTscGet32 (void)`

DESCRIPTION This routine gets a lower half of the 64Bit TSC by RDTSC instruction. RDTSC instruction saves the lower 32Bit in EAX register, so this routine simply returns after executing RDTSC instruction.

RETURNS Lower half of the 64Bit TSC (Timestamp Counter)

SEE ALSO **pentiumALib**

pentiumTscGet64()

| | |
|--------------------|---|
| NAME | <i>pentiumTscGet64()</i> – get 64Bit TSC (Timestamp Counter) |
| SYNOPSIS | <pre>void pentiumTscGet64 () long long int * pTsc; /* Timestamp Counter */</pre> |
| DESCRIPTION | This routine gets 64Bit TSC by RDTSC instruction. Parameter is a pointer of 64Bit variable to store the content of the Counter. |
| RETURNS | N/A |
| SEE ALSO | <i>pentiumALib</i> |

pentiumTscReset()

| | |
|--------------------|--|
| NAME | <i>pentiumTscReset()</i> – reset the TSC (Timestamp Counter) |
| SYNOPSIS | <pre>void pentiumTscReset (void)</pre> |
| DESCRIPTION | This routine resets the TSC by writing zero to the TSC with WRMSR instruction. |
| RETURNS | N/A |
| SEE ALSO | <i>pentiumALib</i> |

period()

| | |
|-----------------|---|
| NAME | <i>period()</i> – spawn a task to call a function periodically |
| SYNOPSIS | <pre>int period (int secs, /* period in seconds */ FUNCPTR func, /* function to call repeatedly */ int arg1, /* first of eight args to pass to func */</pre> |

periodRun()

```

    int    arg2,
    int    arg3,
    int    arg4,
    int    arg5,
    int    arg6,
    int    arg7,
    int    arg8
    )

```

DESCRIPTION This command spawns a task that repeatedly calls a specified function, with up to eight of its arguments, delaying the specified number of seconds between calls.

For example, to have *i()* display task information every 5 seconds, just type:

```
-> period 5, i
```

NOTE The task is spawned using the *sp()* routine. See the description of *sp()* for details about priority, options, stack size, and task ID.

RETURNS A task ID, or ERROR if the task cannot be spawned.

SEE ALSO *usrLib*, *periodRun()*, *sp()*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

periodRun()

NAME *periodRun()* – call a function periodically

SYNOPSIS

```

void periodRun
(
    int    secs, /* no. of seconds to delay between calls */
    FUNCPTR func, /* function to call repeatedly */
    int    arg1, /* first of eight args to pass to func */
    int    arg2,
    int    arg3,
    int    arg4,
    int    arg5,
    int    arg6,
    int    arg7,
    int    arg8
)

```

| | |
|--------------------|---|
| DESCRIPTION | This command repeatedly calls a specified function, with up to eight of its arguments, delaying the specified number of seconds between calls. Normally, this routine is called only by <i>period()</i> , which spawns it as a task. |
| RETURNS | N/A |
| SEE ALSO | <i>usrLib, period(), VxWorks Programmer's Guide: Target Shell</i> |

perror()

| | |
|----------------------|---|
| NAME | <i>perror()</i> – map an error number in errno to an error message (ANSI) |
| SYNOPSIS | <pre>void perror (const char * __s /* error string */)</pre> |
| DESCRIPTION | This routine maps the error number in the integer expression errno to an error message. It writes a sequence of characters to the standard error stream as follows: first (if <i><__s></i> is not a null pointer and the character pointed to by <i><__s></i> is not the null character), the string pointed to by <i><__s></i> followed by a colon (:) and a space; then an appropriate error message string followed by a new-line character. The contents of the error message strings are the same as those returned by <i>strerror()</i> with the argument errno . |
| INCLUDE FILES | stdio.h |
| RETURNS | N/A |
| SEE ALSO | <i>ansiStdio, strerror()</i> |

pfp()

| | |
|-----------------|---|
| NAME | <i>pfp()</i> – return the contents of register pfp (i960) |
| SYNOPSIS | <pre>int pfp (int taskId /* task ID, 0 means default task */)</pre> |

ping()

| | |
|--------------------|--|
| DESCRIPTION | This command extracts the contents of register pfp , the previous frame pointer, from the TCB of a specified task. If <i>taskId</i> is omitted or 0, the current default task is assumed. |
| RETURNS | The contents of the pfp register. |
| SEE ALSO | dbgArchLib , <i>VxWorks Programmer's Guide: Target Shell</i> |

ping()

NAME *ping()* – test that a remote host is reachable

SYNOPSIS

```
STATUS ping
(
    char * host,          /* host to ping */
    int   numPackets,    /* number of packets to receive */
    ulong_t options      /* option flags */
)
```

DESCRIPTION This routine tests that a remote host is reachable by sending ICMP echo request packets, and waiting for replies. It may be called from the VxWorks shell as follows:

```
-> ping "remoteSystem", 1, 0
```

where *remoteSystem* is either a host name that has been previously added to the remote host table by a call to *hostAdd()*, or an Internet address in dot notation (for example, "90.0.0.2").

The second parameter, *numPackets*, specifies the number of ICMP packets to receive from the remote host. If *numPackets* is 1, this routine waits for a single echo reply packet, and then prints a short message indicating whether the remote host is reachable. For all other values of *numPackets*, timing and sequence information is printed as echoed packets are received. If *numPackets* is 0, this routine runs continuously.

If no replies are received within a 5-second timeout period, the routine exits. An ERROR status is returned if no echo replies are received from the remote host.

The following flags may be given through the *options* parameter:

PING_OPT_SILENT

Suppress output. This option is useful for applications that use *ping()* programmatically to examine the return status.

PING_OPT_DONTRROUTE

Do not route packets past the local network.

| | |
|-----------------|--|
| NOTE | The following global variables can be set from the target shell or Windsh to configure the <i>ping()</i> parameters: <i>_pingTxLen</i> Size of the ICMP echo packet (default 64). <i>_pingTxInterval</i> Packet interval in seconds (default 1 second). <i>_pingTxTmo</i> Packet timeout in seconds (default 5 seconds). |
| RETURNS | OK, or ERROR if the remote host is not reachable. |
| ERRNO | EINVAL, S_pingLib_NOT_INITIALIZED, S_pingLib_TIMEOUT |
| SEE ALSO | <i>pingLib</i> |

pingLibInit()

| | |
|--------------------|---|
| NAME | <i>pingLibInit()</i> – initialize the <i>ping()</i> utility |
| SYNOPSIS | <code>STATUS pingLibInit (void)</code> |
| DESCRIPTION | This routine allocates resources used by the <i>ping()</i> utility. It must be called before <i>ping()</i> is used. It is called automatically when the configuration macro <code>INCLUDE_PING</code> is defined. |
| RETURNS | OK, or ERROR if the <i>ping()</i> utility could not be initialized. |
| SEE ALSO | <i>pingLib</i> |

pipeDevCreate()

| | |
|-----------------|---|
| NAME | <i>pipeDevCreate()</i> – create a pipe device |
| SYNOPSIS | <code>STATUS pipeDevCreate (char * name, /* name of pipe to be created */ int nMessages, /* max. number of messages in pipe */</code> |

pipeDrv()

```

    int    nBytes    /* size of each message */
)

```

- DESCRIPTION** This routine creates a pipe device. It allocates memory for the necessary structures and initializes the device. The pipe device will have a maximum of *nMessages* messages of up to *nBytes* each in the pipe at once. When the pipe is full, a task attempting to write to the pipe will be suspended until a message has been read. Messages are lost if written to a full pipe at interrupt level.
- RETURNS** OK, or ERROR if the call fails.
- SEE ALSO** **pipeDrv**

pipeDrv()

- NAME** *pipeDrv()* – initialize the pipe driver
- SYNOPSIS** **STATUS pipeDrv (void)**
- DESCRIPTION** This routine initializes and installs the driver. It must be called before any pipes are created. It is called automatically by the root task, *usrRoot()*, in **usrConfig.c** when the configuration macro **INCLUDE_PIPES** is defined.
- RETURNS** OK, or ERROR if the driver installation fails.
- SEE ALSO** **pipeDrv**

pow()

- NAME** *pow()* – compute the value of a number raised to a specified power (ANSI)
- SYNOPSIS**

```

double pow
(
    double x, /* operand */
    double y /* exponent */
)

```
- DESCRIPTION** This routine returns *x* to the power of *y* in double precision (IEEE double, 53 bits).

A domain error occurs if x is negative and y is not an integral value. A domain error occurs if the result cannot be represented when x is zero and y is less than or equal to zero. A range error may occur.

INCLUDE FILES **math.h**

RETURNS The double-precision value of x to the power of y .

Special cases:

| | | |
|---------------------------------------|----|--------------------------|
| (anything) ** 0 | is | 1 |
| (anything) ** 1 | is | itself |
| (anything) ** NaN | is | NaN |
| NaN ** (anything except 0) | is | NaN |
| +(anything > 1) ** +INF | is | +INF |
| +(anything > 1) ** -INF | is | +0 |
| +(anything < 1) ** +INF | is | +0 |
| +(anything < 1) ** -INF | is | +INF |
| +1 ** +-INF | is | NaN, signal INVALID |
| +0 ** +(anything non-0, NaN) | is | +0 |
| -0 ** +(anything non-0, NaN, odd int) | is | +0 |
| +0 ** -(anything non-0, NaN) | is | +INF, signal DIV-BY-ZERO |
| -0 ** -(anything non-0, NaN, odd int) | is | +INF with signal |
| -0 ** (odd integer) | = | -(+0 ** (odd integer)) |
| +INF ** +(anything except 0, NaN) | is | +INF |
| +INF ** -(anything except 0, NaN) | is | +0 |
| -INF ** (odd integer) | = | -(+INF ** (odd integer)) |
| -INF ** (even integer) | = | (+INF ** (even integer)) |
| -INF ** -(any non-integer, NaN) | is | NaN with signal |
| -(x=anything) ** (k=integer) | is | (-1)**k * (x ** k) |
| -(anything except 0) ** (non-integer) | is | NaN with signal |

SEE ALSO **ansiMath, mathALib**

powf()

NAME *powf()* – compute the value of a number raised to a specified power (ANSI)

SYNOPSIS `float powf`
 (
 `float x, /* operand */`

```
float y /* exponent */  
)
```

- DESCRIPTION** This routine returns the value of x to the power of y in single precision.
- INCLUDE FILES** `math.h`
- RETURNS** The single-precision value of x to the power of y .
- SEE ALSO** `mathALib`

ppc403DevInit()

- NAME** *ppc403DevInit()* – initialize the serial port unit
- SYNOPSIS**
- ```
void ppc403DevInit
(
 PPC403_CHAN * pChan
)
```
- DESCRIPTION** The BSP must already have initialized all the device addresses in the `PPC403_CHAN` structure. This routine initializes some `SIO_CHAN` function pointers and then resets the chip in a quiescent state.
- SEE ALSO** `ppc403Sio`

---

## ***ppc403DummyCallback()***

- NAME** *ppc403DummyCallback()* – dummy callback routine
- SYNOPSIS**
- ```
STATUS ppc403DummyCallback (void)
```
- RETURNS** `ERROR` (always).
- SEE ALSO** `ppc403Sio`

ppc403IntEx()

NAME *ppc403IntEx()* – handle error interrupts

SYNOPSIS

```
void ppc403IntEx
(
    PPC403_CHAN * pChan
)
```

DESCRIPTION This routine handles miscellaneous interrupts on the seial communication controller.

RETURNS N/A

SEE ALSO *ppc403Sio*

ppc403IntRd()

NAME *ppc403IntRd()* – handle a receiver interrupt

SYNOPSIS

```
void ppc403IntRd
(
    PPC403_CHAN * pChan
)
```

DESCRIPTION This routine handles read interrupts from the serial communication controller.

RETURNS N/A

SEE ALSO *ppc403Sio*

ppc403IntWr()

ppc403IntWr()

NAME *ppc403IntWr()* – handle a transmitter interrupt

SYNOPSIS

```
void ppc403IntWr
(
    PPC403_CHAN * pChan
)
```

DESCRIPTION This routine handles write interrupts from the serial communication controller.

RETURNS N/A

SEE ALSO *ppc403Sio*

ppc860DevInit()

NAME *ppc860DevInit()* – initialize the SMC

SYNOPSIS

```
void ppc860DevInit
(
    PPC860SMC_CHAN * pChan
)
```

DESCRIPTION This routine is called to initialize the chip to a quiescent state. Note that the **smcNum** field of **PPC860SMC_CHAN** must be either 1 or 2.

SEE ALSO *ppc860Sio*

ppc860Int()

NAME *ppc860Int()* – handle an SMC interrupt

SYNOPSIS

```
void ppc860Int
(
    PPC860SMC_CHAN * pChan
)
```

DESCRIPTION This routine is called to handle SMC interrupts.

SEE ALSO **ppc860Sio**

2

pppDelete()

NAME *pppDelete()* – delete a PPP network interface

SYNOPSIS

```
void pppDelete
(
    int unit /* PPP interface unit number to delete */
)
```

DESCRIPTION This routine deletes the Point-to-Point Protocol (PPP) network interface specified by the unit number *unit*.

A Link Control Protocol (LCP) terminate request packet is sent to notify the peer of the impending PPP link shut-down. The associated serial interface (*tty*) is then detached from the PPP driver, and the PPP interface is deleted from the list of network interfaces. Finally, all resources associated with the PPP link are returned to the VxWorks system.

RETURNS N/A

SEE ALSO **pppLib**

pppHookAdd()

NAME *pppHookAdd()* – add a hook routine on a unit basis

SYNOPSIS

```
STATUS pppHookAdd
(
    int    unit,    /* unit number */
    FUNCPTR hookRtn, /* hook routine */
    int    hookType /* hook type connect/disconnect */
)
```

DESCRIPTION This routine adds a hook to the Point-to-Point Protocol (PPP) channel. The parameters to this routine specify the unit number (*unit*) of the PPP interface, the hook routine (*hookRtn*),

pppHookDelete()

and the type of hook specifying either a connect hook or a disconnect hook (*hookType*). The following hook types can be specified for the *hookType* parameter:

PPP_HOOK_CONNECT

Specify a connect hook.

PPP_HOOK_DISCONNECT

Specify a disconnect hook.

RETURNS OK, or ERROR if the hook cannot be added to the unit.

SEE ALSO *pppHookLib*, *pppHookDelete()*

pppHookDelete()

NAME *pppHookDelete()* – delete a hook routine on a unit basis

SYNOPSIS

```
STATUS pppHookDelete
(
    int unit,      /* unit number */
    int hookType /* hook type connect/disconnect */
)
```

DESCRIPTION This routine deletes a hook added previously to the Point-to-Point Protocol (PPP) channel. The parameters to this routine specify the unit number (*unit*) of the PPP interface and the type of hook specifying either a connect hook or a disconnect hook (*hookType*). The following hook types can be specified for the *hookType* parameter:

PPP_HOOK_CONNECT

Specify a connect hook.

PPP_HOOK_DISCONNECT

Specify a disconnect hook.

RETURNS OK, or ERROR if the hook cannot be deleted for the unit.

SEE ALSO *pppHookLib*, *pppHookAdd()*

pppInfoGet()

| | |
|--------------------|---|
| NAME | <i>pppInfoGet()</i> – get PPP link status information |
| SYNOPSIS | <pre>STATUS pppInfoGet (int unit, /* PPP interface unit number to examine */ PPP_INFO * pInfo /* PPP_INFO structure to be filled */)</pre> |
| DESCRIPTION | <p>This routine gets status information pertaining to the specified Point-to-Point Protocol (PPP) link, regardless of the link state. State and option information is gathered for the Link Control Protocol (LCP), Internet Protocol Control Protocol (IPCP), Password Authentication Protocol (PAP), and Challenge-Handshake Authentication Protocol (CHAP).</p> <p>The PPP link information is returned through a PPP_INFO structure, which is defined in h/netinet/ppp/pppShow.h.</p> |
| RETURNS | OK, or ERROR if <i>unit</i> is an invalid PPP unit number. |
| SEE ALSO | pppShow , pppLib |

pppInfoShow()

| | |
|--------------------|---|
| NAME | <i>pppInfoShow()</i> – display PPP link status information |
| SYNOPSIS | <pre>void pppInfoShow (void)</pre> |
| DESCRIPTION | <p>This routine displays status information pertaining to each initialized Point-to-Point Protocol (PPP) link, regardless of the link state. State and option information is gathered for the Link Control Protocol (LCP), Internet Protocol Control Protocol (IPCP), Password Authentication Protocol (PAP), and Challenge-Handshake Authentication Protocol (CHAP).</p> |
| RETURNS | N/A |
| SEE ALSO | pppShow , pppLib |

pppInit()

pppInit()**NAME** *pppInit()* – initialize a PPP network interface

SYNOPSIS

```
int pppInit
(
    int          unit,          /* PPP interface unit number to initialize */
    char *      devname,       /* name of the tty device to be used */
    char *      local_addr,    /* local IP address of the PPP interface */
    char *      remote_addr,   /* remote peer IP address of the PPP link */
    int         baud,          /* baud rate of tty; NULL = default */
    PPP_OPTIONS * pOptions,    /* PPP options structure pointer */
    char *      fOptions       /* PPP options file name */
)
```

DESCRIPTION This routine initializes a Point-to-Point Protocol (PPP) network interface. The parameters to this routine specify the unit number (*unit*) of the PPP interface, the name of the serial interface (*tty*) device (*devname*), the IP addresses of the local and remote ends of the link, the interface baud rate, an optional configuration options structure pointer, and an optional configuration options file name.

IP ADDRESSES The *local_addr* and *remote_addr* parameters specify the IP addresses of the local and remote ends of the PPP link, respectively. If *local_addr* is NULL, the local IP address will be negotiated with the remote peer. If the remote peer does not assign a local IP address, it will default to the address associated with the local target's machine name. If *remote_addr* is NULL, the remote peer's IP address will be obtained from the remote peer. A routing table entry to the remote peer will be automatically added once the PPP link is established.

CONFIGURATION OPTIONS STRUCTURE

The optional parameter *pOptions* specifies configuration options for the PPP link. If NULL, this parameter is ignored, otherwise it is assumed to be a pointer to a PPP_OPTIONS options structure (defined in **h/netinet/ppp/options.h**).

The "flags" member of the PPP_OPTIONS structure is a bit-mask, where the following bit-flags may be specified:

OPT_NO_ALL

Do not request/allow any options.

OPT_PASSIVE_MODE

Set passive mode.

OPT_SILENT_MODE

Set silent mode.

- OPT_DEFAULTROUTE**
Add default route.
- OPT_PROXYARP**
Add proxy ARP entry.
- OPT_IPCP_ACCEPT_LOCAL**
Accept peer's idea of the local IP address.
- OPT_IPCP_ACCEPT_REMOTE**
Accept peer's idea of the remote IP address.
- OPT_NO_IP**
Disable IP address negotiation.
- OPT_NO_ACC**
Disable address/control compression.
- OPT_NO_PC**
Disable protocol field compression.
- OPT_NO_VJ**
Disable VJ (Van Jacobson) compression.
- OPT_NO_VJCCOMP**
Disable VJ (Van Jacobson) connection ID compression.
- OPT_NO_ASYNCMAP**
Disable async map negotiation.
- OPT_NO_MN**
Disable magic number negotiation.
- OPT_NO_MRU**
Disable MRU (Maximum Receive Unit) negotiation.
- OPT_NO_PAP**
Do not allow PAP authentication with peer.
- OPT_NO_CHAP**
Do not allow CHAP authentication with peer.
- OPT_REQUIRE_PAP**
Require PAP authentication with peer.
- OPT_REQUIRE_CHAP**
Require CHAP authentication with peer.
- OPT_LOGIN**
Use the login password database for PAP authentication of peer.
- OPT_DEBUG**
Enable PPP daemon debug mode.
- OPT_DRIVER_DEBUG**
Enable PPP driver debug mode.

pppInit()

The remaining members of the **PPP_OPTIONS** structure specify PPP configurations options that require string values. These options are:

char *asyncmap

Set the desired async map to the specified string.

char *escape_chars

Set the chars to escape on transmission to the specified string.

char *vj_max_slots

Set maximum number of VJ compression header slots to the specified string.

char *netmask

Set netmask value for negotiation to the specified string.

char *mru

Set MRU value for negotiation to the specified string.

char *mtu

Set MTU (Maximum Transmission Unit) value for negotiation to the specified string.

char *lcp_echo_failure

Set the maximum number of consecutive LCP echo failures to the specified string.

char *lcp_echo_interval

Set the interval in seconds between LCP echo requests to the specified string.

char *lcp_restart

Set the timeout in seconds for the LCP negotiation to the specified string.

char *lcp_max_terminate

Set the maximum number of transmissions for LCP termination requests to the specified string.

char *lcp_max_configure

Set the maximum number of transmissions for LCP configuration requests to the specified string.

char *lcp_max_failure

Set the maximum number of LCP configuration NAKs to the specified string.

char *ipcp_restart

Set the timeout in seconds for IPCP negotiation to the specified string.

char *ipcp_max_terminate

Set the maximum number of transmissions for IPCP termination requests to the specified string.

char *ipcp_max_configure

Set the maximum number of transmissions for IPCP configuration requests to the specified string.

- char *ipcp_max_failure**
Set the maximum number of IPCP configuration NAKs to the specified string.
- char *local_auth_name**
Set the local name for authentication to the specified string.
- char *remote_auth_name**
Set the remote name for authentication to the specified string.
- char *pap_file**
Get PAP secrets from the specified file. This option is necessary if either peer requires PAP authentication.
- char *pap_user_name**
Set the user name for PAP authentication with the peer to the specified string.
- char *pap_passwd**
Set the password for PAP authentication with the peer to the specified string.
- char *pap_restart**
Set the timeout in seconds for PAP negotiation to the specified string.
- char *pap_max_authreq**
Set the maximum number of transmissions for PAP authentication requests to the specified string.
- char *chap_file**
Get CHAP secrets from the specified file. This option is necessary if either peer requires CHAP authentication.
- char *chap_restart**
Set the timeout in seconds for CHAP negotiation to the specified string.
- char *chap_interval**
Set the interval in seconds for CHAP rechallenge to the specified string.
- char *chap_max_challenge**
Set the maximum number of transmissions for CHAP challenge to the specified string.

CONFIGURATION OPTIONS FILE

The optional parameter *fOptions* specifies configuration options for the PPP link. If NULL, this parameter is ignored, otherwise it is assumed to be the name of a configuration options file. The format of the options file is one option per line; comment lines start with "#". The following options are recognized:

- no_all**
Do not request/allow any options.
- passive_mode**
Set passive mode.

pppInit()

silent_mode

Set silent mode.

defaultroute

Add default route.

proxyarp

Add proxy ARP entry.

ipcp_accept_local

Accept peer's idea of the local IP address.

ipcp_accept_remote

Accept peer's idea of the remote IP address.

no_ip

Disable IP address negotiation.

no_acc

Disable address/control compression.

no_pc

Disable protocol field compression.

no_vj

Disable VJ (Van Jacobson) compression.

no_vjcomp

Disable VJ (Van Jacobson) connection ID compression.

no_asyncmap

Disable async map negotiation.

no_mn

Disable magic number negotiation.

no_mru

Disable MRU (Maximum Receive Unit) negotiation.

no_pap

Do not allow PAP authentication with peer.

no_chap

Do not allow CHAP authentication with peer.

require_pap

Require PAP authentication with peer.

require_chap

Require CHAP authentication with peer.

login

Use the login password database for PAP authentication of peer.

debug

Enable PPP daemon debug mode.

driver_debug

Enable PPP driver debug mode.

asyncmap value

Set the desired async map to the specified value.

escape_chars value

Set the chars to escape on transmission to the specified value.

vj_max_slots value

Set maximum number of VJ compression header slots to the specified value.

netmask value

Set netmask value for negotiation to the specified value.

mru value

Set MRU value for negotiation to the specified value.

mtu value

Set MTU value for negotiation to the specified value.

lcp_echo_failure value

Set the maximum consecutive LCP echo failures to the specified value.

lcp_echo_interval value

Set the interval in seconds between LCP echo requests to the specified value.

lcp_restart value

Set the timeout in seconds for the LCP negotiation to the specified value.

lcp_max_terminate value

Set the maximum number of transmissions for LCP termination requests to the specified value.

lcp_max_configure value

Set the maximum number of transmissions for LCP configuration requests to the specified value.

lcp_max_failure value

Set the maximum number of LCP configuration NAKs to the specified value.

ipcp_restart value

Set the timeout in seconds for IPCP negotiation to the specified value.

ipcp_max_terminate value

Set the maximum number of transmissions for IPCP termination requests to the specified value.

ipcp_max_configure value

Set the maximum number of transmissions for IPCP configuration requests to the

pppInit()

specified value.

ipcp_max_failure *value*

Set the maximum number of IPCP configuration NAKs to the specified value.

local_auth_name *name*

Set the local name for authentication to the specified name.

remote_auth_name *name*

Set the remote name for authentication to the specified name.

pap_file *file*

Get PAP secrets from the specified file. This option is necessary if either peer requires PAP authentication.

pap_user_name *name*

Set the user name for PAP authentication with the peer to the specified name.

-

Set the password for PAP authentication with the peer to the specified password.

pap_restart *value*

Set the timeout in seconds for PAP negotiation to the specified value.

pap_max_authreq *value*

Set the maximum number of transmissions for PAP authentication requests to the specified value.

chap_file *file*

Get CHAP secrets from the specified file. This option is necessary if either peer requires CHAP authentication.

chap_restart *value*

Set the timeout in seconds for CHAP negotiation to the specified value.

chap_interval *value*

Set the interval in seconds for CHAP rechallenge to the specified value.

chap_max_challenge *value*

Set the maximum number of transmissions for CHAP challenge to the specified value.

AUTHENTICATION The VxWorks PPP implementation supports two separate user authentication protocols: the Password Authentication Protocol (PAP) and the Challenge-Handshake Authentication Protocol (CHAP). If authentication is required by either peer, it must be satisfactorily completed before the PPP link becomes fully operational. If authentication fails, the link will be automatically terminated.

EXAMPLES The following routine initializes a PPP interface that uses the target's second serial port (*/tyCo/1*). The local IP address is 90.0.0.1; the IP address of the remote peer is 90.0.0.10.

The baud rate is the default rate for the *tty* device. VJ compression and authentication have been disabled, and LCP echo requests have been enabled.

```
PPP_OPTIONS pppOpt; /* PPP configuration options */
void routine ()
{
    pppOpt.flags = OPT_PASSIVE_MODE | OPT_NO_PAP | OPT_NO_CHAP | OPT_NO_VJ;
    pppOpt.lcp_echo_interval = "30";
    pppOpt.lcp_echo_failure = "10";
    pppInit (0, "/tyCo/1", "90.0.0.1", "90.0.0.10", 0, &pppOpt, NULL);
}
```

The following routine generates the same results as the previous example. The difference is that the configuration options are obtained from a file rather than a structure.

```
pppFile = "phobos:/tmp/ppp_options"; /* PPP configuration options file */
void routine ()
{
    pppInit (0, "/tyCo/1", "90.0.0.1", "90.0.0.10", 0, NULL, pppFile);
}
```

where `phobos:/tmp/ppp_options` contains:

```
passive
no_pap
no_chap
no_vj
lcp_echo_interval 30
lcp_echo_failure 10
```

RETURNS OK, or ERROR if the PPP interface cannot be initialized because the daemon task cannot be spawned or memory is insufficient.

SEE ALSO `pppLib`, `pppShow`, `pppDelete()`, *VxWorks Programmer's Guide: Network*

pppSecretAdd()

NAME `pppSecretAdd()` – add a secret to the PPP authentication secrets table

SYNOPSIS

```
STATUS pppSecretAdd
(
    char * client, /* client being authenticated */
    char * server, /* server performing authentication */
    char * secret, /* secret used for authentication */
```

pppSecretDelete()

```

char * addr  /* acceptable client IP addresses */
)

```

DESCRIPTION This routine adds a secret to the Point-to-Point Protocol (PPP) authentication secrets table. This table may be used by the Password Authentication Protocol (PAP) and Challenge-Handshake Authentication Protocol (CHAP) user authentication protocols.

When a PPP link is established, a "server" may require a "client" to authenticate itself using a "secret". Clients and servers obtain authentication secrets by searching secrets files, or by searching the secrets table constructed by this routine. Clients and servers search the secrets table by matching client and server names with table entries, and retrieving the associated secret.

Client and server names in the table consisting of "*" are considered wildcards; they serve as matches for any client and/or server name if an exact match cannot be found.

If *secret* starts with "@", *secret* is assumed to be the name of a file, wherein the actual secret can be read.

If *addr* is not NULL, it should contain a list of acceptable client IP addresses. When a server is authenticating a client and the client's IP address is not contained in the list of acceptable addresses, the link is terminated. Any IP address will be considered acceptable if *addr* is NULL. If this parameter is "-", all IP addresses are disallowed.

RETURNS OK, or ERROR if the secret cannot be added to the table.

SEE ALSO `pppSecretLib`, `pppSecretDelete()`, `pppSecretShow()`

pppSecretDelete()

NAME `pppSecretDelete()` – delete a secret from the PPP authentication secrets table

SYNOPSIS

```

STATUS pppSecretDelete
(
char * client, /* client being authenticated */
char * server, /* server performing authentication */
char * secret /* secret used for authentication */
)

```

DESCRIPTION This routine deletes a secret from the Point-to-Point Protocol (PPP) authentication secrets table. When searching for a secret to delete from the table, the wildcard substitution (using "*") is not performed for client and/or server names. The *client*, *server*, and *secret* strings must match the table entry exactly in order to be deleted.

RETURNS OK, or ERROR if the table entry being deleted is not found.

SEE ALSO `pppSecretLib`, `pppSecretAdd()`, `pppSecretShow()`

pppSecretShow()

NAME `pppSecretShow()` – display the PPP authentication secrets table

SYNOPSIS `void pppSecretShow (void)`

DESCRIPTION This routine displays the Point-to-Point Protocol (PPP) authentication secrets table. The information in the secrets table may be used by the Password Authentication Protocol (PAP) and Challenge-Handshake Authentication Protocol (CHAP) user authentication protocols.

RETURNS N/A

SEE ALSO `pppShow`, `pppLib`, `pppSecretAdd()`, `pppSecretDelete()`

pppstatGet()

NAME `pppstatGet()` – get PPP link statistics

SYNOPSIS `STATUS pppstatGet`
`(`
`int unit, /* PPP interface unit number to examine */`
`PPP_STAT * pStat /* PPP_STAT structure to be filled */`
`)`

DESCRIPTION This routine gets statistics for the specified Point-to-Point Protocol (PPP) link. Detailed are the numbers of bytes and packets received and sent through the PPP interface.

The PPP link statistics are returned through a `PPP_STAT` structure, which is defined in `h/netinet/ppp/pppShow.h`.

RETURNS OK, or ERROR if `unit` is an invalid PPP unit number.

SEE ALSO `pppShow`, `pppLib`

pppstatShow()

| | |
|--------------------|---|
| NAME | <i>pppstatShow()</i> – display PPP link statistics |
| SYNOPSIS | <code>void pppstatShow (void)</code> |
| DESCRIPTION | This routine displays statistics for each initialized Point-to-Point Protocol (PPP) link. Detailed are the numbers of bytes and packets received and sent through each PPP interface. |
| RETURNS | N/A |
| SEE ALSO | pppShow , pppLib |

printErr()

| | |
|--------------------|--|
| NAME | <i>printErr()</i> – write a formatted string to the standard error stream |
| SYNOPSIS | <pre>int printErr (const char * fmt /* format string to write */)</pre> |
| DESCRIPTION | This routine writes a formatted string to standard error. Its function and syntax are otherwise identical to <i>printf()</i> . |
| RETURNS | The number of characters output, or ERROR if there is an error during output. |
| SEE ALSO | fioLib , printf() |

printErrno()

| | |
|--------------------|--|
| NAME | <i>printErrno()</i> – print the definition of a specified error status value |
| SYNOPSIS | <pre>void printErrno (int errNo /* status code whose name is to be printed */)</pre> |
| DESCRIPTION | <p>This command displays the error-status string, corresponding to a specified error-status value. It is only useful if the error-status symbol table has been built and included in the system. If <i>errNo</i> is zero, then the current task status is used by calling <i>errnoGet()</i>.</p> <p>This facility is described in errnoLib.</p> |
| RETURNS | N/A |
| SEE ALSO | usrLib , errnoLib , <i>errnoGet()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , windsh , <i>Tornado User's Guide: Shell</i> |

printf()

| | |
|--------------------|--|
| NAME | <i>printf()</i> – write a formatted string to the standard output stream (ANSI) |
| SYNOPSIS | <pre>int printf (const char * fmt /* format string to write */)</pre> |
| DESCRIPTION | <p>This routine writes output to standard output under control of the string <i>fmt</i>. The string <i>fmt</i> contains ordinary characters, which are written unchanged, plus conversion specifications, which cause the arguments that follow <i>fmt</i> to be converted and printed as part of the formatted string.</p> <p>The number of arguments for the format is arbitrary, but they must correspond to the conversion specifications in <i>fmt</i>. If there are insufficient arguments, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The routine returns when the end of the format string is encountered.</p> <p>The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: ordinary multibyte characters</p> |

printf()

(not %) that are copied unchanged to the output stream; and conversion specification, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the % character. After the %, the following appear in sequence:

- Zero or more flags (in any order) that modify the meaning of the conversion specification.
- An optional minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces (by default) on the left (or right, if the left adjustment flag, described later, has been given) to the field width. The field width takes the form of an asterisk (*) (described later) or a decimal integer.
- An optional precision that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, and **X** conversions, the number of digits to appear after the decimal-point character for **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversions, or the maximum number of characters to be written from a string in the **s** conversion. The precision takes the form of a period (.) followed either by an asterisk (*) (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional **h** specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **short int** or **unsigned short int** argument (the argument will have been promoted according to the integral promotions, and its value converted to **short int** or **unsigned short int** before printing); an optional **h** specifying that a following **n** conversion specifier applies to a pointer to a **short int** argument; an optional **l** (el) specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **long int** or **unsigned long int** argument; or an optional **l** specifying that a following **n** conversion specifier applies to a pointer to a **long int** argument. If an **h** or **l** appears with any other conversion specifier, the behavior is undefined.
- WARNING: ANSI C also specifies an optional **L** in some of the same contexts as **l** above, corresponding to a **long double** argument. However, the current release of the VxWorks libraries does not support **long double** data; using the optional **L** gives unpredictable results.
- A character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both, can be indicated by an asterisk (*). In this case, an **int** argument supplies the field width or precision. The arguments specifying field width, or precision, or both, should appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

The flag characters and their meanings are:

-

The result of the conversion will be left-justified within the field. (it will be

right-justified if this flag is not specified.)

+

The result of a signed conversion will always begin with a plus or minus sign. (It will begin with a sign only when a negative value is converted if this flag is not specified.)

space

If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space will be prefixed to the result. If the **space** and **+** flags both appear, the **space** flag will be ignored.

#

The result is to be converted to an "alternate form." For **o** conversion it increases the precision to force the first digit of the result to be a zero. For **x** (or **X**) conversion, a non-zero result will have "0x" (or "0X") prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if no digit follows it). For **g** and **G** conversions, trailing zeros will not be removed from the result. For other conversions, the behavior is undefined.

0

For **d**, **i**, **o**, **u**, **x**, **X**, **e**, **E**, **f**, **g**, and **G** conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the **0** and **-** flags both appear, the **0** flag will be ignored. For **d**, **i**, **o**, **u**, **x**, and **X** conversions, if a precision is specified, the **0** flag will be ignored. For other conversions, the behavior is undefined.

The conversion specifiers and their meanings are:

d, i

The **int** argument is converted to signed decimal in the style **[-]dddd**. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

o, u, x, X

The **unsigned int** argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x** or **X**) in the style **dddd**; the letters abcdef are used for **x** conversion and the letters ABCDEF for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

f

The **double** argument is converted to decimal notation in the style **[-]ddd.ddd**, where the number of digits after the decimal point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and

printf()

the # flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.

e, E

The **double** argument is converted in the style [-]d.ddde+/-dd, where there is one digit before the decimal-point character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The E conversion specifier will produce a number with E instead of e, introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

g, G

The **double** argument is converted in style f or e (or in style E in the case of a G conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as 1. The style used depends on the value converted; style e (or E) will be used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a decimal-point character appears only if it is followed by a digit.

c

The **int** argument is converted to an **unsigned char**, and the resulting character is written.

s

The argument should be a pointer to an array of character type. Characters from the array are written up to (but not including) a terminating null character; if the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array will contain a null character.

p

The argument should be a pointer to **void**. The value of the pointer is converted to a sequence of printable characters, in hexadecimal representation (prefixed with "0x").

n

The argument should be a pointer to an integer into which the number of characters written to the output stream so far by this call to *fprintf()* is written. No argument is converted.

%

A % is written. No argument is converted. The complete conversion specification is %%.

If a conversion specification is invalid, the behavior is undefined.

If any argument is, or points to, a union or an aggregate (except for an array of character type using **s** conversion, or a pointer using **p** conversion), the behavior is undefined.

In no case does a non-existent or small field width cause truncation of a field if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

| | |
|----------------------|---|
| INCLUDE FILES | fiolib.h |
| RETURNS | The number of characters written, or a negative value if an output error occurs. |
| SEE ALSO | fiolib , fprintf() , <i>American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: Input/Output (stdio.h)</i> |

printLogo()

| | |
|--------------------|--|
| NAME | <i>printLogo()</i> – print the VxWorks logo |
| SYNOPSIS | <code>void printLogo (void)</code> |
| DESCRIPTION | This command displays the VxWorks banner seen at boot time. It also displays the VxWorks version number and kernel version number. |
| RETURNS | N/A |
| SEE ALSO | usrLib , <i>VxWorks Programmer's Guide: Target Shell</i> , windsh , <i>Tornado User's Guide: Shell</i> |

proxyArpLibInit()

| | |
|--------------------|---|
| NAME | <i>proxyArpLibInit()</i> – initialize proxy ARP |
| SYNOPSIS | <pre>STATUS proxyArpLibInit (int clientSizeLog2, /* client table size as power of two */ int portSizeLog2 /* port table size as power of two */)</pre> |
| DESCRIPTION | This routine initializes the proxy ARP library by initializing tables and structures and adding the hooks to process ARP, proxy messages, and broadcasts. <i>clientSizeLog2</i> |

proxyNetCreate()

specifies the client hash table size as a power of two. *portSizeLog2* specifies the port hash table as a power of two. If either of these parameters is zero, a default value will be used. By default, *proxyArpLibInit()* enables broadcast forwarding of the BOOTP server port.

This routine should be called only once; subsequent calls have no effect.

RETURNS OK, or ERROR if unsuccessful.

SEE ALSO *proxyArpLib*

proxyNetCreate()

NAME *proxyNetCreate()* – create a proxy ARP network

SYNOPSIS

```
STATUS proxyNetCreate
(
    char * proxyAddr, /* proxy network address */
    char * mainAddr  /* main network address */
)
```

DESCRIPTION This routine creates a proxy network with the interface *proxyAddr* as the proxy network and the interface *mainAddr* as the main network. The interfaces and the routing tables must be set up correctly, prior to calling this routine. That is, the interfaces must be attached, addresses must be set, and there should be a network route to *mainAddr* and no routes to *proxyAddr*.

proxyAddr and *mainAddr* must reside in the same network address space.

RETURNS OK, or ERROR if unsuccessful.

ERRNO S_proxyArpLib_INVALID_INTERFACE
S_proxyArpLib_INVALID_ADDRESS

SEE ALSO *proxyArpLib*

proxyNetDelete()

NAME *proxyNetDelete()* – delete a proxy network

SYNOPSIS `STATUS proxyNetDelete`
`(`
`char * proxyAddr /* proxy net address */`
`)`

DESCRIPTION This routine deletes the proxy network specified by *proxyAddr*. It also removes all the proxy clients that exist on that network.

RETURNS OK, or ERROR if unsuccessful.

SEE ALSO `proxyArpLib`

proxyNetShow()

NAME *proxyNetShow()* – show proxy ARP networks

SYNOPSIS `void proxyNetShow (void)`

DESCRIPTION This routine displays the proxy networks and their associated clients.

EXAMPLE `-> proxyNetShow`
`main interface 147.11.1.182 proxy interface 147.11.1.183`
`client 147.11.1.184`

RETURNS N/A

SEE ALSO `proxyArpLib`

proxyPortFwdOff()

| | |
|--------------------|--|
| NAME | <i>proxyPortFwdOff()</i> – disable broadcast forwarding for a particular port |
| SYNOPSIS | <pre>STATUS proxyPortFwdOff (int port /* port number */)</pre> |
| DESCRIPTION | This routine disables broadcast forwarding on port number <i>port</i> . To disable the (previously enabled) forwarding of all ports via <i>proxyPortFwdOn()</i> , specify zero for <i>port</i> . |
| RETURNS | OK, or ERROR if unsuccessful. |
| SEE ALSO | <i>proxyArpLib</i> |

proxyPortFwdOn()

| | |
|--------------------|---|
| NAME | <i>proxyPortFwdOn()</i> – enable broadcast forwarding for a particular port |
| SYNOPSIS | <pre>STATUS proxyPortFwdOn (int port /* port number */)</pre> |
| DESCRIPTION | This routine enables broadcasts destined for the port, <i>port</i> , to be forwarded to and from the proxy network. To enable all ports, specify zero for <i>port</i> . |
| RETURNS | OK, or ERROR if unsuccessful. |
| SEE ALSO | <i>proxyArpLib</i> |

proxyPortShow()

| | |
|-----------------|---|
| NAME | <i>proxyPortShow()</i> – show enabled ports |
| SYNOPSIS | <pre>void proxyPortShow (void)</pre> |

| | |
|--------------------|---|
| DESCRIPTION | This routine displays the ports currently enabled. |
| EXAMPLE | <pre>-> proxyPortShow enabled ports: port 67</pre> |
| RETURNS | N/A |
| SEE ALSO | proxyArpLib |

proxyReg()

| | |
|--------------------|---|
| NAME | <i>proxyReg()</i> – register a proxy client |
| SYNOPSIS | <pre>STATUS proxyReg (char * ifName, /* interface name */ char * proxyAddr /* proxy address */)</pre> |
| DESCRIPTION | This routine sends a message over the network interface <i>ifName</i> to register <i>proxyAddr</i> as a proxy client. |
| RETURNS | OK, or ERROR if unsuccessful. |
| SEE ALSO | proxyLib |

proxyUnreg()

| | |
|--------------------|---|
| NAME | <i>proxyUnreg()</i> – unregister a proxy client |
| SYNOPSIS | <pre>STATUS proxyUnreg (char * ifName, /* interface name */ char * proxyAddr /* proxy address */)</pre> |
| DESCRIPTION | This routine sends a message over the network interface <i>ifName</i> to unregister <i>proxyAddr</i> as a proxy client. |

psr()

RETURNS OK, or ERROR if unsuccessful.

SEE ALSO **proxyLib**

psr()

NAME *psr()* – return the contents of the processor status register (SPARC)

SYNOPSIS

```
int psr
(
    int taskId /* task ID, 0 means default task */
)
```

DESCRIPTION This command extracts the contents of the processor status register from the TCB of a specified task. If *taskId* is omitted or 0, the default task is assumed.

RETURNS The contents of the processor status register.

SEE ALSO **dbgArchLib**, **psrShow()**, *VxWorks Programmer's Guide: Target Shell*

psrShow()

NAME *psrShow()* – display the meaning of a specified **psr** value, symbolically (SPARC)

SYNOPSIS

```
void psrShow
(
    ULONG psrValue /* psr value to show */
)
```

DESCRIPTION This routine displays the meaning of all the fields in a specified **psr** value, symbolically. Extracted from **psl.h**:

Definition of bits in the Sun-4 PSR (Processor Status Register)

```
-----
| IMPL | VER |      ICC      | resvd | EC | EF | PIL | S | PS | ET | CWP |
|      |     | N | Z | V | C |     |   |   |   |   |   |   |
|-----|-----|---|---|---|-----|-----|-----|-----|-----|-----|
| 31  28  27  24  23  22  21  20  19  14  13  12  11  8  7  6  5  4  0
```

For compatibility with future revisions, reserved bits are defined to be initialized to zero and, if written, must be preserved.

```

EXAMPLE          -> psrShow 0x00001FE7
                    Implementation 0, mask version 0:
                    Fujitsu MB86900 or LSI L64801, 7 windows
                      no SWAP, FSQRT, CP, extended fp instructions
                    Condition codes: . . . .
                    Coprocessor enables: . EF
                    Processor interrupt level: f
                    Flags: S PS ET
                    Current window pointer: 0x07
                    ->

```

RETURNS N/A

SEE ALSO **dbgArchLib**, *psr()*, *SPARC Architecture Manual*

psrShow()

NAME *psrShow()* – display the meaning of a specified PSR value, symbolically (ARM)

SYNOPSIS `STATUS psrShow`
`(`
 `UINT32 psrval /* psr value to show */`
`)`

DESCRIPTION This routine displays the meaning of all fields in a specified PSR value, symbolically.

RETURNS OK, always.

SEE ALSO **dbgArchLib**

ptyDevCreate()

NAME *ptyDevCreate()* – create a pseudo terminal

SYNOPSIS

```
STATUS ptyDevCreate
(
    char * name,      /* name of pseudo terminal */
    int   rdBufSize, /* size of terminal read buffer */
    int   wrtBufSize /* size of write buffer */
)
```

DESCRIPTION This routine creates a master and slave device which can then be opened by the master and slave processes. The master process simulates the "hardware" side of the driver, while the slave process is the application program that normally talks to a tty driver. Data written to the master device can then be read on the slave device, and vice versa.

RETURNS OK, or ERROR if memory is insufficient.

SEE ALSO *ptyDrv*

ptyDrv()

NAME *ptyDrv()* – initialize the pseudo-terminal driver

SYNOPSIS STATUS *ptyDrv* (void)

DESCRIPTION This routine initializes the pseudo-terminal driver. It must be called before any other routine in this module.

RETURNS OK, or ERROR if the master or slave devices cannot be installed.

SEE ALSO *ptyDrv*

putc()

NAME *putc()* – write a character to a stream (ANSI)

SYNOPSIS

```
int putc
(
    int    c, /* character to write */
    FILE * fp /* stream to write to */
)
```

DESCRIPTION This routine writes a character *c* to a specified stream, at the position indicated by the stream's file position indicator (if defined), and advances the indicator appropriately. This routine is equivalent to *fputc()*, except that if it is implemented as a macro, it may evaluate *fp* more than once; thus, the argument should never be an expression with side effects.

INCLUDE FILES `stdio.h`

RETURNS The character written, or EOF if a write error occurs, with the error indicator set for the stream.

SEE ALSO `ansiStdio`, *fputc()*

putchar()

NAME *putchar()* – write a character to the standard output stream (ANSI)

SYNOPSIS

```
int putchar
(
    int c /* character to write */
)
```

DESCRIPTION This routine writes a character *c* to the standard output stream, at the position indicated by the stream's file position indicator (if defined), and advances the indicator appropriately.

This routine is equivalent to *putc()* with a second argument of `stdout`.

INCLUDE FILES `stdio.h`

putenv()

RETURNS The character written, or EOF if a write error occurs, with the error indicator set for the standard output stream.

SEE ALSO *ansiStdio*, *putc()*, *fputc()*

putenv()

NAME *putenv()* – set an environment variable

SYNOPSIS

```
STATUS putenv
(
    char * pEnvString /* string to add to env */
)
```

DESCRIPTION This routine sets an environment variable to a value by altering an existing variable or creating a new one. The parameter points to a string of the form "variableName=value". Unlike the UNIX implementation, the string is copied to a private buffer.

RETURNS OK, or ERROR if space cannot be malloc'd.

SEE ALSO *envLibInit()*, *getenv()*

puts()

NAME *puts()* – write a string to the standard output stream (ANSI)

SYNOPSIS

```
int puts
(
    char const * s /* string to write */
)
```

DESCRIPTION This routine writes to the standard output stream a specified string *s*, minus the terminating null character, and appends a new-line character to the output.

INCLUDE FILES *stdio.h*

RETURNS A non-negative value, or EOF if a write error occurs.

SEE ALSO *ansiStdio*, *fputs()*

putw()

NAME *putw()* – write a word (32-bit integer) to a stream

SYNOPSIS

```
int putw
(
    int    w, /* word (32-bit integer) */
    FILE * fp /* output stream */
)
```

DESCRIPTION This routine appends the 32-bit quantity *w* to a specified stream.
This routine is provided for compatibility with earlier VxWorks releases.

INCLUDE FILES *stdio.h*

RETURNS The value written.

SEE ALSO *ansiStdio*

pwd()

NAME *pwd()* – print the current default directory

SYNOPSIS `void pwd (void)`

DESCRIPTION This command displays the current working device/directory.

RETURNS N/A

SEE ALSO *usrLib*, *cd()*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

qsort()

qsort()**NAME** `qsort()` – sort an array of objects (ANSI)

SYNOPSIS

```
void qsort
(
    void *                bot, /* initial element in array */
    size_t                nmemb, /* no. of objects in array */
    size_t                size, /* size of array element */
    int (* compar) (const void * ,
    const void *          ) /* comparison function */
)
```

DESCRIPTION This routine sorts an array of *nmemb* objects, the initial element of which is pointed to by *bot*. The size of each object is specified by *size*.

The contents of the array are sorted into ascending order according to a comparison function pointed to by *compar*, which is called with two arguments that point to the objects being compared. The function shall return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

If two elements compare as equal, their order in the sorted array is unspecified.

INCLUDE FILES `stdlib.h`**RETURNS** N/A**SEE ALSO** `ansiStdlib`

r0()**NAME** `r0()` – return the contents of register **r0** (also **r1** – **r14**) (ARM)

SYNOPSIS

```
int r0
(
    int taskId /* task ID, 0 means default task */
)
```

DESCRIPTION This command extracts the contents of register **r0** from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.

Similar routines are provided for registers (**r1** – **r14**): *r1()* – *r14()*.

RETURNS The contents of register **r0** (or the requested register).

SEE ALSO **dbgArchLib**, *VxWorks Programmer's Guide: Debugging*

r3()

NAME *r3()* – return the contents of register **r3** (also **r4** – **r15**) (i960)

SYNOPSIS

```
int r3
(
    int taskId /* task ID, 0 means default task */
)
```

DESCRIPTION This command extracts the contents of register **r3** from the TCB of a specified task. If *taskId* is omitted or 0, the current default task is assumed.

Routines are provided for all local registers (**r3** – **r15**): *r3()* – *r15()*.

RETURNS The contents of the **r3** register (or the requested register).

SEE ALSO **dbgArchLib**, *VxWorks Programmer's Guide: Target Shell*

raise()

NAME *raise()* – send a signal to the caller's task

SYNOPSIS

```
int raise
(
    int signo /* signal to send to caller's task */
)
```

DESCRIPTION This routine sends the signal *signo* to the task invoking the call.

RETURNS OK (0), or ERROR (-1) if the signal number or task ID is invalid.

ERRNO EINVAL

SEE ALSO **sigLib**

ramDevCreate()

NAME *ramDevCreate()* – create a RAM disk device

SYNOPSIS

```
BLK_DEV *ramDevCreate
(
    char * ramAddr,      /* where it is in memory (0 = malloc) */
    int   bytesPerBlk,  /* number of bytes per block */
    int   blksPerTrack, /* number of blocks per track */
    int   nBlocks,     /* number of blocks on this device */
    int   blkOffset    /* no. of blks to skip at start of device */
)
```

DESCRIPTION This routine creates a RAM disk device.

Memory for the RAM disk can be pre-allocated separately; if so, the *ramAddr* parameter should be the address of the pre-allocated device memory. Or, memory can be automatically allocated with *malloc()* by setting *ramAddr* to zero.

The *bytesPerBlk* parameter specifies the size of each logical block on the RAM disk. If *bytesPerBlk* is zero, 512 is used.

The *blksPerTrack* parameter specifies the number of blocks on each logical track of the RAM disk. If *blksPerTrack* is zero, the count of blocks per track is set to *nBlocks* (i.e., the disk is defined as having only one track).

The *nBlocks* parameter specifies the size of the disk, in blocks. If *nBlocks* is zero, a default size is used. The default is calculated using a total disk size of either 51,200 bytes or one-half of the size of the largest memory area available, whichever is less. This default disk size is then divided by *bytesPerBlk* to determine the number of blocks.

The *blkOffset* parameter specifies an offset, in blocks, from the start of the device to be used when writing or reading the RAM disk. This offset is added to the block numbers passed by the file system during disk accesses. (VxWorks file systems always use block numbers beginning at zero for the start of a device.) This offset value is typically useful only if a specific address is given for *ramAddr*. Normally, *blkOffset* is 0.

FILE SYSTEMS Once the device has been created, it must be associated with a name and a file system (dosFs, rt11Fs, or rawFs). This is accomplished using the file system's device initialization routine or make-file-system routine, e.g., *dosFsDevInit()* or *dosFsMkfs()*. The *ramDevCreate()* call returns a pointer to a block device structure (BLK_DEV). This structure contains fields that describe the physical properties of a disk device and specify the addresses of routines within the **ramDrv** driver. The BLK_DEV structure address must be passed to the desired file system (dosFs, rt11Fs or rawFs) via the file system's device initialization or make-file-system routine. Only then is a name and file system associated with the device, making it available for use.

EXAMPLE

In the following example, a 200-Kbyte RAM disk is created with automatically allocated memory, 512-byte blocks, a single track, and no block offset. The device is then initialized for use with dosFs and assigned the name "DEV1:":

```
BLK_DEV *pBlkDev;
DOS_VOL_DESC *pVolDesc;
pBlkDev = ramDevCreate (0, 512, 400, 400, 0);
pVolDesc = dosFsMkfs ("DEV1:", pBlkDev);
```

The *dosFsMkfs()* routine calls *dosFsDevInit()* with default parameters and initializes the file system on the disk by calling *ioctl()* with the **FIODISKINIT** function.

If the RAM disk memory already contains a disk image created elsewhere, the first argument to *ramDevCreate()* should be the address in memory, and the formatting parameters -- *bytesPerBlk*, *blksPerTrack*, *nBlocks*, and *blkOffset* -- must be identical to those used when the image was created. For example:

```
pBlkDev = ramDevCreate (0xc0000, 512, 400, 400, 0);
pVolDesc = dosFsDevInit ("DEV1:", pBlkDev, NULL);
```

In this case, *dosFsDevInit()* must be used instead of *dosFsMkfs()*, because the file system already exists on the disk and should not be re-initialized. This procedure is useful if a RAM disk is to be created at the same address used in a previous boot of VxWorks. The contents of the RAM disk will then be preserved.

These same procedures apply when creating a RAM disk with rt11Fs using *rt11FsDevInit()* and *rt11FsMkfs()*, or creating a RAM disk with rawFs using *rawFsDevInit()*.

RETURNS

A pointer to a block device structure (**BLK_DEV**) or **NULL** if memory cannot be allocated for the device structure or for the RAM disk.

SEE ALSO

ramDrv, *dosFsMkfs()*, *dosFsDevInit()*, *rt11FsDevInit()*, *rt11FsMkfs()*, *rawFsDevInit()*

ramDrv()

NAME

ramDrv() – prepare a RAM disk driver for use (optional)

SYNOPSIS

```
STATUS ramDrv (void)
```

DESCRIPTION

This routine performs no real function, except to provide compatibility with earlier versions of **ramDrv** and to parallel the initialization function found in true disk device drivers. It also is used in **usrConfig.c** to link in the RAM disk driver when building VxWorks. Otherwise, there is no need to call this routine before using the RAM disk driver.

rand()

RETURNS OK, always.

SEE ALSO **ramDrv**

rand()

NAME *rand()* – generate a pseudo-random integer between 0 and **RAND_MAX** (ANSI)

SYNOPSIS `int rand (void)`

DESCRIPTION This routine generates a pseudo-random integer between 0 and **RAND_MAX**. The seed value for *rand()* can be reset with *srand()*.

INCLUDE FILES **stdlib.h**

RETURNS A pseudo-random integer.

SEE ALSO **ansiStdlib**, *srand()*

rawFsDevInit()

NAME *rawFsDevInit()* – associate a block device with raw volume functions

SYNOPSIS `RAW_VOL_DESC *rawFsDevInit`
`(`
`char * volName, /* volume name */`
`BLK_DEV * pBlkDev /* pointer to block device info */`
`)`

DESCRIPTION This routine takes a block device created by a device driver and defines it as a raw file system volume. As a result, when high-level I/O operations, such as *open()* and *write()*, are performed on the device, the calls will be routed through **rawFsLib**.

This routine associates *volName* with a device and installs it in the VxWorks I/O System's device table. The driver number used when the device is added to the table is that which was assigned to the raw library during *rawFsInit()*. (The driver number is kept in the global variable **rawFsDrvNum**.)

The **BLK_DEV** structure specified by *pBlkDev* contains configuration data describing the device and the addresses of five routines which will be called to read blocks, write blocks,

reset the device, check device status, and perform other control functions (*ioctl()*). These routines will not be called until they are required by subsequent I/O operations.

RETURNS A pointer to the volume descriptor (**RAW_VOL_DESC**), or NULL if there is an error.

SEE ALSO **rawFsLib**

rawFsInit()

NAME *rawFsInit()* – prepare to use the raw volume library

SYNOPSIS

```
STATUS rawFsInit
(
    int maxFiles /* max no. of simultaneously open files */
)
```

DESCRIPTION This routine initializes the raw volume library. It must be called exactly once, before any other routine in the library. The argument specifies the number of file descriptors that may be open at once. This routine allocates and sets up the necessary memory structures and initializes semaphores.

This routine also installs raw volume library routines in the VxWorks I/O system driver table. The driver number assigned to **rawFsLib** is placed in the global variable **rawFsDrvNum**. This number will later be associated with system file descriptors opened to rawFs devices.

This initialization is enabled when the configuration macro **INCLUDE_RAWFS** is defined; *rawFsInit()* is then called from the root task, *usrRoot()*, in *usrConfig.c*.

RETURNS OK or ERROR.

SEE ALSO **rawFsLib**

rawFsModeChange()

NAME *rawFsModeChange()* – modify the mode of a raw device volume

SYNOPSIS

```
void rawFsModeChange
(
    RAW_VOL_DESC * vdptr, /* pointer to volume descriptor */
)
```

```
int          newMode /* O_RDONLY/O_WRONLY/O_RDWR (both) */  
)
```

DESCRIPTION This routine sets the device's mode to *newMode* by setting the mode field in the **BLK_DEV** structure. This routine should be called whenever the read and write capabilities are determined, usually after a ready change.

The driver's device initialization routine should initially set the mode to **O_RDWR** (i.e., both **O_RDONLY** and **O_WRONLY**).

RETURNS N/A

SEE ALSO **rawFsLib**, ***rawFsReadyChange()***

rawFsReadyChange()

NAME ***rawFsReadyChange()*** – notify **rawFsLib** of a change in ready status

SYNOPSIS

```
void rawFsReadyChange  
(  
    RAW_VOL_DESC * vdptr /* pointer to volume descriptor */  
)
```

DESCRIPTION This routine sets the volume descriptor state to **RAW_VD_READY_CHANGED**. It should be called whenever a driver senses that a device has come on-line or gone off-line, (e.g., a disk has been inserted or removed).

After this routine has been called, the next attempt to use the volume will result in an attempted remount.

RETURNS N/A

SEE ALSO **rawFsLib**

rawFsVolUnmount()

NAME *rawFsVolUnmount()* – disable a raw device volume

SYNOPSIS

```
STATUS rawFsVolUnmount
(
    RAW_VOL_DESC * vdptr /* pointer to volume descriptor */
)
```

DESCRIPTION This routine is called when I/O operations on a volume are to be discontinued. This is commonly done before changing removable disks. All buffered data for the volume is written to the device (if possible), any open file descriptors are marked as obsolete, and the volume is marked as not mounted.

Because this routine will flush data from memory to the physical device, it should not be used in situations where the disk-change is not recognized until after a new disk has been inserted. In these circumstances, use the ready-change mechanism. (See the manual entry for *rawFsReadyChange()*.)

This routine may also be called by issuing an *ioctl()* call using the **FIOUNMOUNT** function code.

RETURNS OK, or ERROR if the routine cannot access the volume.

SEE ALSO *rawFsLib*, *rawFsReadyChange()*

rcmd()

NAME *rcmd()* – execute a shell command on a remote machine

SYNOPSIS

```
int rcmd
(
    char * host,          /* host name or inet address */
    int  remotePort,    /* remote port to connect to (rshd) */
    char * localUser,   /* local user name */
    char * remoteUser,  /* remote user name */
    char * cmd,         /* command */
    int * fd2p          /* if this pointer is non-zero, stderr socket is */
                        /* and socket descriptor is filled in */
)
```

read()

- DESCRIPTION** This routine executes a command on a remote machine, using the remote shell daemon, *rshd*, on the remote system. It is analogous to the UNIX routine *rcmd()*.
- RETURNS** A socket descriptor if the remote shell daemon accepts, or ERROR if the remote command fails.
- SEE ALSO** *remLib*, UNIX BSD 4.3 manual entry for *rcmd()*

read()

NAME *read()* – read bytes from a file or device

SYNOPSIS

```
int read
(
    int fd,          /* file descriptor from which to read */
    char * buffer,   /* pointer to buffer to receive bytes */
    size_t maxbytes /* max no. of bytes to read into buffer */
)
```

DESCRIPTION This routine reads a number of bytes (less than or equal to *maxbytes*) from a specified file descriptor and places them in *buffer*. It calls the device driver to do the work.

RETURNS The number of bytes read (between 1 and *maxbytes*, 0 if end of file), or ERROR if the file descriptor does not exist, the driver does not have a read routines, or the driver returns ERROR. If the driver does not have a read routine, *errno* is set to ENOTSUP.

SEE ALSO *ioLib*

readdir()

NAME *readdir()* – read one entry from a directory (POSIX)

SYNOPSIS

```
struct dirent *readdir
(
    DIR * pDir /* pointer to directory descriptor */
)
```

| | |
|--------------------|---|
| DESCRIPTION | <p>This routine obtains directory entry data for the next file from an open directory. The <i>pDir</i> parameter is the pointer to a directory descriptor (DIR) which was returned by a previous <i>opendir()</i>.</p> <p>This routine returns a pointer to a dirent structure which contains the name of the next file. Empty directory entries and MS-DOS volume label entries are not reported. The name of the file (or subdirectory) described by the directory entry is returned in the d_name field of the dirent structure. The name is a single null-terminated string.</p> <p>The returned dirent pointer will be NULL, if it is at the end of the directory or if an error occurred. Because there are two conditions which might cause NULL to be returned, the task's error number (errno) must be used to determine if there was an actual error. Before calling <i>readdir()</i>, set errno to OK. If a NULL pointer is returned, check the new value of errno. If errno is still OK, the end of the directory was reached; if not, errno contains the error code for an actual error which occurred.</p> |
| RETURNS | A pointer to a dirent structure, or NULL if there is an end-of-directory marker or error. |
| SEE ALSO | <i>dirLib</i> , <i>opendir()</i> , <i>closedir()</i> , <i>rewinddir()</i> , <i>ls()</i> |

realloc()

| | |
|--------------------|--|
| NAME | <i>realloc()</i> – reallocate a block of memory (ANSI) |
| SYNOPSIS | <pre>void *realloc (void * pBlock, /* block to reallocate */ size_t newSize /* new block size */)</pre> |
| DESCRIPTION | <p>This routine changes the size of a specified block of memory and returns a pointer to the new block of memory. The contents that fit inside the new size (or old size if smaller) remain unchanged. The memory alignment of the new block is not guaranteed to be the same as the original block.</p> |
| RETURNS | A pointer to the new block of memory, or NULL if the call fails. |
| SEE ALSO | memLib , <i>American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: General Utilities (stdlib.h)</i> |

reboot()

NAME *reboot()* – reset network devices and transfer control to boot ROMs

SYNOPSIS

```
void reboot
(
    int startType /* how the boot ROMS will reboot */
)
```

DESCRIPTION This routine returns control to the boot ROMs after calling a series of preliminary shutdown routines that have been added via *rebootHookAdd()*, including routines to reset all network devices. After calling the shutdown routines, interrupts are locked, all caches are cleared, and control is transferred to the boot ROMs.

The bit values for *startType* are defined in **sysLib.h**:

BOOT_NORMAL (0x00)

causes the system to go through the countdown sequence and try to reboot VxWorks automatically. Memory is not cleared.

BOOT_NO_AUTOBOOT (0x01)

causes the system to display the VxWorks boot prompt and wait for user input to the boot ROM monitor. Memory is not cleared.

BOOT_CLEAR (0x02)

the same as **BOOT_NORMAL**, except that memory is cleared.

BOOT_QUICK_AUTOBOOT (0x04)

the same as **BOOT_NORMAL**, except the countdown is shorter.

RETURNS N/A

SEE ALSO *rebootLib*, *sysToMonitor()*, *rebootHookAdd()*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

rebootHookAdd()

NAME *rebootHookAdd()* – add a routine to be called at reboot

SYNOPSIS

```
STATUS rebootHookAdd
(
    FUNCPTR rebootHook /* routine to be called at reboot */
)
```

DESCRIPTION This routine adds the specified routine to a list of routines to be called when VxWorks is rebooted. The specified routine should be declared as follows:

```
void rebootHook
(
    int startType /* startType is passed to all hooks */
)
```

RETURNS OK, or ERROR if memory is insufficient.

SEE ALSO *rebootLib*, *reboot()*

recv()

NAME *recv()* – receive data from a socket

SYNOPSIS

```
int recv
(
    int s, /* socket to receive data from */
    char * buf, /* buffer to write data to */
    int bufLen, /* length of buffer */
    int flags /* flags to underlying protocols */
)
```

DESCRIPTION This routine receives data from a connection-based (stream) socket.

The maximum length of *buf* is subject to the limits on TCP buffer size; see the discussion of `SO_RCVBUF` in the *setsockopt()* manual entry.

You may OR the following values into the *flags* parameter with this operation:

MSG_OOB (0x1)
Out-of-band data.

MSG_PEEK (0x2)
Return data without removing it from socket.

RETURNS The number of bytes received, or ERROR if the call fails.

SEE ALSO *sockLib*, *setsockopt()*

recvfrom()

NAME *recvfrom()* – receive a message from a socket

SYNOPSIS

```
int recvfrom
(
    int          s,          /* socket to receive from */
    char *       buf,       /* pointer to data buffer */
    int         bufLen,     /* length of buffer */
    int         flags,     /* flags to underlying protocols */
    struct sockaddr * from, /* where to copy sender's addr */
    int *       pFromLen /* value/result length of from */
)
```

DESCRIPTION This routine receives a message from a datagram socket regardless of whether it is connected. If *from* is non-zero, the address of the sender's socket is copied to it. The value-result parameter *pFromLen* should be initialized to the size of the *from* buffer. On return, *pFromLen* contains the actual size of the address stored in *from*.

The maximum length of *buf* is subject to the limits on UDP buffer size; see the discussion of `SO_RCVBUF` in the *setsockopt()* manual entry.

You may OR the following values into the *flags* parameter with this operation:

`MSG_OOB` (0x1) Out-of-band data.

`MSG_PEEK` (0x2) Return data without removing it from socket.

RETURNS The number of number of bytes received, or `ERROR` if the call fails.

SEE ALSO *sockLib*, *setsockopt()*

recvmsg()

NAME *recvmsg()* – receive a message from a socket

SYNOPSIS

```
int recvmsg
(
    int          sd, /* socket to receive from */
    struct msghdr * mp, /* scatter-gather message header */
    int         flags /* flags to underlying protocols */
)
```

| | |
|--------------------|---|
| DESCRIPTION | This routine receives a message from a datagram socket. It may be used in place of <i>recvfrom()</i> to decrease the overhead of breaking down the message-header structure <i>msg_hdr</i> for each message. For BSD 4.4 sockets a copy of the <i>mp>msg_iov</i> array will be made. This requires a cluster from the network stack system pool of <i>size mp>msg_iovlen * sizeof (struct iovec)</i> or 8 bytes. |
| RETURNS | The number of bytes received, or ERROR if the call fails. |
| SEE ALSO | <i>sockLib</i> |

reld()

| | |
|--------------------|---|
| NAME | <i>reld()</i> – reload an object module |
| SYNOPSIS | <pre> MODULE_ID reld (void * nameOrId, /* name or ID of the object module file */ int options /* options, currently unused */) </pre> |
| DESCRIPTION | <p>This routine unloads a specified object module from the system, and then calls <i>ld()</i> to load a new copy of the same name.</p> <p>If the file was originally loaded using a complete pathname, then <i>reld()</i> will use the complete name to locate the file. If the file was originally loaded using a partial pathname, then the current working directory must be changed to the working directory in use at the time of the original load.</p> |
| RETURNS | A module ID (type <i>MODULE_ID</i>), or NULL. |
| SEE ALSO | <i>unldLib</i> , <i>unld()</i> |

remCurIdGet()

NAME *remCurIdGet()* – get the current user name and password

SYNOPSIS

```
void remCurIdGet
(
    char * user, /* where to return current user name */
    char * passwd /* where to return current password */
)
```

DESCRIPTION This routine gets the user name and password currently used for remote host access privileges and copies them to *user* and *passwd*. Either parameter can be initialized to NULL, and the corresponding item will not be passed.

RETURNS N/A

SEE ALSO *remLib*, *iam()*, *whoami()*

remCurIdSet()

NAME *remCurIdSet()* – set the remote user name and password

SYNOPSIS

```
STATUS remCurIdSet
(
    char * newUser, /* user name to use on remote */
    char * newPasswd /* password to use on remote (NULL = none) */
)
```

DESCRIPTION This routine specifies the user name that will have access privileges on the remote machine. The user name must exist in the remote machine's */etc/passwd*, and if it has been assigned a password, the password must be specified in *newPasswd*. Either parameter can be NULL, and the corresponding item will not be set.

The maximum length of the user name and the password is **MAX_IDENTITY_LEN**(defined in *remLib.h*).

NOTE A more convenient version of this routine is *iam()*, intended for use from the shell.

RETURNS OK, or ERROR if the name or password is too long.

SEE ALSO *remLib*, *iam()*, *whoami()*

remove()

| | |
|--------------------|---|
| NAME | <i>remove()</i> – remove a file (ANSI) |
| SYNOPSIS | <pre>STATUS remove (const char * name /* name of the file to remove */)</pre> |
| DESCRIPTION | This routine deletes a specified file. It calls the driver for the particular device on which the file is located to do the work. |
| RETURNS | OK if there is no delete routine for the device or the driver returns OK; ERROR if there is no such device or the driver returns ERROR. |
| SEE ALSO | <i>ioLib</i> , <i>American National Standard for Information Systems – Programming Language – C</i> , <i>ANSI X3.159-1989: Input/Output (stdio.h)</i> , |

rename()

| | |
|--------------------|---|
| NAME | <i>rename()</i> – change the name of a file |
| SYNOPSIS | <pre>int rename (const char * oldname, /* name of file to rename */ const char * newname /* name with which to rename file */)</pre> |
| DESCRIPTION | This routine changes the name of a file from <i>oldfile</i> to <i>newfile</i> . |
| NOTE | Only certain devices support <i>rename()</i> . To confirm that your device supports it, consult the respective <i>xxDrv</i> or <i>xxFs</i> listings to verify that ioctl <i>FIORENAME</i> exists. For example, <i>dosFs</i> and <i>rt11Fs</i> support <i>rename()</i> , but <i>netDrv</i> and <i>nfsDrv</i> do not. |
| RETURNS | OK, or ERROR if the file could not be opened or renamed. |
| SEE ALSO | <i>ioLib</i> |

repeat()

| | |
|--------------------|---|
| NAME | <i>repeat()</i> – spawn a task to call a function repeatedly |
| SYNOPSIS | <pre>int repeat (int n, /* no. of times to call func (0=forever) */ FUNCPTR func, /* function to call repeatedly */ int arg1, /* first of eight args to pass to func */ int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8)</pre> |
| DESCRIPTION | This command spawns a task that calls a specified function <i>n</i> times, with up to eight of its arguments. If <i>n</i> is 0, the routine is called endlessly, or until the spawned task is deleted. |
| NOTE | The task is spawned using <i>sp()</i> . See the description of <i>sp()</i> for details about priority, options, stack size, and task ID. |
| RETURNS | A task ID, or ERROR if the task cannot be spawned. |
| SEE ALSO | <i>usrLib</i> , <i>repeatRun()</i> , <i>sp()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , <i>windsh</i> , <i>Tornado User's Guide: Shell</i> |

repeatRun()

| | |
|-----------------|---|
| NAME | <i>repeatRun()</i> – call a function repeatedly |
| SYNOPSIS | <pre>void repeatRun (int n, /* no. of times to call func (0=forever) */ FUNCPTR func, /* function to call repeatedly */ int arg1, /* first of eight args to pass to func */ int arg2, int arg3,</pre> |

```

int    arg4,
int    arg5,
int    arg6,
int    arg7,
int    arg8
)

```

- DESCRIPTION** This command calls a specified function *n* times, with up to eight of its arguments. If *n* is 0, the routine is called endlessly.
- Normally, this routine is called only by *repeat()*, which spawns it as a task.
- RETURNS** N/A
- SEE ALSO** *usrLib*, *repeat()*, *VxWorks Programmer's Guide: Target Shell*

resolvDNComp()

- NAME** *resolvDNComp()* – compress a DNS name in a DNS packet
- SYNOPSIS**
- ```

int resolvDNComp
(
 const u_char * exp_dn, /* ptr to the expanded domain name */
 u_char * comp_dn, /* ptr to where to output the compressed name */
 int length, /* length of the buffer pointed by comp_dn */
 u_char * * dnptrs, /* ptr to a ptr list of compressed names */
 u_char * * lastdnptr /* ptr to the last entry pointed by dnptrs */
)

```
- DESCRIPTION** This routine takes the expanded domain name referenced in the *exp\_dn* parameter, compresses it, and stores the compressed name in the location pointed to by the *comp\_dn* parameter. The *length* parameter passes in the length of the buffer starting at *comp\_dn*. The *dnptrs* parameter is a pointer to a list of pointers to previously compressed names. The *lastdnptr* parameter points to the last entry in the *dnptrs* array.
- RETURNS** The size of the compressed name, or ERROR.
- SEE ALSO** *resolvLib*, *resolvGetHostByName()*, *resolvGetHostByAddr()*, *resolvDNExpand()*, *resolvInit()*, *resolvSend()*, *resolvParamsSet()*, *resolvParamsGet()*, *resolvMkQuery()*, *resolvQuery()*

---

## *resolvDNExpand()*

**NAME** *resolvDNExpand()* – expand a DNS compressed name from a DNS packet

**SYNOPSIS**

```
int resolvDNExpand
(
 const u_char * msg, /* ptr to the start of the DNS packet */
 const u_char * eomorig, /* ptr to last location +1 of the DNS packet */
 const u_char * comp_dn, /* ptr to the compressed domain name */
 u_char * exp_dn, /* ptr to where the expanded DN is output */
 int length /* length of the buffer pointed by exp_dn */
)
```

**DESCRIPTION** This function expands a compressed DNS name from a DNS packet. The *msg* parameter points to that start of the DNS packet. The *eomorig* parameter points to the last location of the DNS packet plus 1. The *comp\_dn* parameter points to the compressed domain name, and *exp\_dn* parameter expects a pointer to a buffer. Upon function completion, this buffer contains the expanded domain name. Use the *length* parameter to pass in the size of the buffer referenced by the *exp\_dn* parameter.

**RETURNS** The length of the expanded domain name, or ERROR on failure.

**SEE ALSO** *resolvLib*, *resolvGetHostByName()*, *resolvGetHostByAddr()*, *resolvInit()*, *resolvDNComp()*, *resolvSend()*, *resolvParamsSet()*, *resolvParamsGet()*, *resolvMkQuery()*, *resolvQuery()*

---

## *resolvGetHostByAddr()*

**NAME** *resolvGetHostByAddr()* – query the DNS server for the host name of an IP address

**SYNOPSIS**

```
struct hostent * resolvGetHostByAddr
(
 const char * pInetAddr,
 char * pHostBuf,
 int bufLen
)
```

**DESCRIPTION** This function returns a **hostent** structure, which is defined as follows:

```
struct hostent
{
```

```

char * h_name; /* official name of host */
char ** h_aliases; /* alias list */
int h_addrtype; /* address type */
int h_length; /* length of address */
char ** h_addr_list; /* list of addresses from name server */
unsigned int h_ttl; /* Time to Live in Seconds for this entry */
}

```

The `h_aliases` and `h_addr_type` vectors are NULL-terminated.

The `pinetAddr` parameter passes in the IP address (in network byte order) for the host whose name you want to discover. The `pBuf` and `bufLen` parameters specify the location and size (512 bytes or more) of the buffer that is to receive the `hostent` structure. `resolvGetHostByAddr()` returns host addresses are returned in network byte order.

**RETURNS**

A pointer to a `hostent` structure if the host is found, or NULL if the parameters are invalid, host is not found, or the buffer is too small.

**ERRNO**

```

S_resolvLib_INVALID_PARAMETER
S_resolvLib_BUFFER_2_SMALL
S_resolvLib_TRY_AGAIN
S_resolvLib_HOST_NOT_FOUND
S_resolvLib_NO_DATA
S_resolvLib_NO_RECOVERY

```

**SEE ALSO**

`resolvLib`, `resolvGetHostByName()`, `resolvInit()`, `resolvDNExpand()`, `resolvDNComp()`, `resolvSend()`, `resolvParamsSet()`, `resolvParamsGet()`, `resolvMkQuery()`, `resolvQuery()`

---

## *resolvGetHostByName()*

**NAME**

`resolvGetHostByName()` – query the DNS server for the IP address of a host

**SYNOPSIS**

```

struct hostent * resolvGetHostByName
(
 char * pHostName, /* ptr to the name of the host */
 char * pHostBuf, /* ptr to the buffer used by hostent structure */
 int bufLen /* length of the buffer */
)

```

**DESCRIPTION**

This function returns a `hostent` structure. This structure is defined as follows:

```

struct hostent
{
 char * h_name; /* official name of host */

```

**resolvInit()**

```

char ** h_aliases; /* alias list */
int h_addrtype; /* address type */
int h_length; /* length of address */
char ** h_addr_list; /* list of addresses from name server */
unsigned int h_ttl; /* Time to Live in Seconds for this entry */
}

```

The **h\_aliases** and **h\_addr\_type** vectors are NULL-terminated.

Specify the host you want to query in *pHostname*. Use *pBuf* and *bufLen* to specify the location and size of a buffer to receive the **hostent** structure and its associated contents. Host addresses are returned in network byte order. Given the information this routine retrieves, the *pBuf* buffer should be 512 bytes or larger.

- RETURNS** A pointer to a **hostent** structure if the host is found, or NULL if the parameters are invalid, the host is not found, or the buffer is too small.
- ERRNO** S\_resolvLib\_INVALID\_PARAMETER  
S\_resolvLib\_BUFFER\_2\_SMALL  
S\_resolvLib\_TRY\_AGAIN  
S\_resolvLib\_HOST\_NOT\_FOUND  
S\_resolvLib\_NO\_DATA  
S\_resolvLib\_NO\_RECOVERY
- SEE ALSO** *resolvLib*, *resolvInit()*, *resolvGetHostByAddr()*, *resolvDNExpand()*, *resolvDNComp()*, *resolvSend()*, *resolvParamsSet()*, *resolvParamsGet()*, *resolvMkQuery()*, *resolvQuery()*

---

**resolvInit()**

- NAME** *resolvInit()* – initialize the resolver library
- SYNOPSIS**
- ```

STATUS resolvInit
(
    char * pNameServer,      /* pointer to Name server IP address */
    char * pDefaultDomainName, /* default domain name */
    FUNCPTR pdnsDebugRtn    /* function ptr to debug routine */
)

```
- DESCRIPTION** This function initializes the resolver. *pNameServer* is a single IP address for a name server in dotted decimal notation. *pDefaultDomainName* is the default domain name to be appended to names without a dot. The function pointer *pdnsDebugRtn* is set to the resolver debug function. Additional name servers can be configured using the function *resolvParamsSet()*.

RETURNS OK or ERROR.

SEE ALSO *resolvLib*, *resolvGetHostByName()*, *resolvGetHostByAddr()*, *resolvDNExpand()*, *resolvDNComp()*, *resolvSend()*, *resolvParamsSet()*, *resolvParamsGet()*, *resolvQuery()*

resolvMkQuery()

NAME *resolvMkQuery()* – create all types of DNS queries

SYNOPSIS

```
int resolvMkQuery
(
    int          op,          /* set to desire query QUERY or IQUERY */
    const char * dname,      /* domain name to be use in the query */
    int          class,      /* query class for IP is C_IN */
    int          type,       /* type is T_A, T_PTR, ... */
    const char * data,       /* resource Record (RR) data */
    int          datalen,    /* length of the RR */
    const char * newrr_in,   /* not used always set to NULL */
    char *       buf,        /* out of the constructed query */
    int          buflen      /* length of the buffer for the query */
)
```

DESCRIPTION This routine uses the input parameters to create a domain name query. You can set the *op* parameter to QUERY or IQUERY. Specify the domain name in *dname*, the class in *class*, the query type in *type*. Valid values for type include T_A, T_PTR, and so on. Use *data* to add Resource Record data to the query. Use *datalen* to pass in the length of the data buffer. Set *newrr_in* to NULL. This parameter is reserved for future use. The *buf* parameter expects a pointer to the output buffer for the constructed query. Use *buflen* to pass in the length of the buffer referenced in *buf*.

RETURNS The length of the constructed query or ERROR.

SEE ALSO *resolvLib*, *resolvGetHostByName()*, *resolvGetHostByAddr()*, *resolvDNExpand()*, *resolvDNComp()*, *resolvSend()*, *resolvParamsSet()*, *resolvParamsGet()*, *resolvInit()*, *resolvQuery()*

resolvParamsGet()

NAME *resolvParamsGet()* – get the parameters which control the resolver library

SYNOPSIS

```
void resolvParamsGet
(
    RESOLV_PARAMS_S * pResolvParams /* ptr to resolver parameter struct */
)
```

DESCRIPTION This routine copies the resolver parameters to the **RESOLV_PARAMS_S** structure referenced in the *pResolvParams* parameter. The **RESOLV_PARAMS_S** structure is defined in **resolvLib.h** as follows:

```
typedef struct
{
    char    queryOrder;
    char    domainName [MAXDNAME];
    char    nameServersAddr [MAXNS][MAXIPADDRLLEN];
} RESOLV_PARAMS_S;
```

Typically, you call this function just before calling *resolvParamsSet()*. The *resolvParamsGet()* call populates the **RESOLV_PARAMS_S** structure. You can then modify the default values just before calling *resolvParamsSet()*.

RETURNS N/A

SEE ALSO *resolvLib*, *resolvGetHostByName()*, *resolvGetHostByAddr()*, *resolvDNExpand()*, *resolvDNComp()*, *resolvSend()*, *resolvParamsSet()*, *resolvInit()*, *resolvMkQuery()*, *resolvQuery()*

resolvParamsSet()

NAME *resolvParamsSet()* – set the parameters which control the resolver library

SYNOPSIS

```
STATUS resolvParamsSet
(
    RESOLV_PARAMS_S * pResolvParams /* ptr to resolver parameter struct */
)
```

DESCRIPTION This routine sets the resolver parameters. *pResolvParams* passes in a pointer to a **RESOLV_PARAMS_S** structure, which is defined as follows:


```
typedef struct
{
    char    queryOrder;
    char    domainName [MAXDNAME];
    char    nameServersAddr [MAXNS][MAXIPADDRLLEN];
} RESOLV_PARAMS_S;
```

Use the members of this structure to specify the settings you want to apply to the resolver. It is important to remember that multiple tasks can use the resolver library and that the settings specified in this `RESOLV_PARAMS_S` structure affect all queries from all tasks. In addition, you should set resolver parameters at initialization and not while queries could be in progress. Otherwise, the results of the query are unpredictable.

Before calling *resolvParamsSet()*, you should first call *resolvParamsGet()* to populate a `RESOLV_PARAMS_S` structure with the current settings. Then you change the values of the members that interest you.

Valid values for the `queryOrder` member of `RESOLV_PARAMS_S` structure are defined in `resolvLib.h`. Set the `domainName` member to the domain to which this resolver belongs. Set the `nameServersAddr` member to the IP addresses of the DNS server that the resolver can query. You must specify the IP addresses in standard dotted decimal notation. This function tries to validate the values in the `queryOrder` and `nameServerAddr` members. This function does not try to validate the domain name.

RETURNS OK if the parameters are valid, ERROR otherwise.

SEE ALSO *resolvLib*, *resolvGetHostByName()*, *resolvGetHostByAddr()*, *resolvDNExpand()*, *resolvDNComp()*, *resolvSend()*, *resolvInit()*, *resolvParamsGet()*, *resolvMkQuery()*, *resolvQuery()*

resolvQuery()

NAME *resolvQuery()* – construct a query, send it, wait for a response

SYNOPSIS

```
int resolvQuery
(
    char *    name,    /* domain name */
    int      class,   /* query class for IP is C_IN */
    int      type,    /* type is T_A, T_PTR, ... */
    u_char * answer, /* buffer to put answer */
    int      anslen  /* length of answer buffer */
)
```

resolvSend()

| | |
|--------------------|--|
| DESCRIPTION | This routine constructs a query for the domain specified in the <i>name</i> parameter. The <i>class</i> parameter specifies the class of the query. The <i>type</i> parameter specifies the type of query. The routine then sends the query to the DNS server. When the server responds, the response is validated and copied to the buffer you supplied in the <i>answer</i> parameter. Use the <i>anslen</i> parameter to pass in the size of the buffer referenced in <i>answer</i> . |
| RETURNS | The length of the response or ERROR. |
| ERRNO | S_resolvLib_TRY_AGAIN S_resolvLib_HOST_NOT_FOUND S_resolvLib_NO_DATA S_resolvLib_NO_RECOVERY |
| SEE ALSO | resolvLib, resolvGetHostByName(), resolvGetHostByAddr(), resolvDNExpand(), resolvDNComp(), resolvInit(), resolvParamsSet(), resolvParamsGet(), resolvMkQuery() |

resolvSend()

| | |
|--------------------|---|
| NAME | <i>resolvSend()</i> – send a pre-formatted query and return the answer |
| SYNOPSIS | <pre>int resolvSend (const char * buf, /* pre-formatted query */ int buflen, /* length of query */ char * answer, /* buffer for answer */ int anslen /* length of answer */)</pre> |
| DESCRIPTION | This routine takes a pre-formatted DNS query and sends it to the domain server. Use <i>buf</i> to pass in a pointer to the query. Use <i>buflen</i> to pass in the size of the buffer referenced in <i>buf</i> . The <i>answer</i> parameter expects a pointer to a buffer into which this routine can write the answer retrieved from the server. Use <i>anslen</i> to pass in the size of the buffer you have provided in <i>anslen</i> . |
| RETURNS | The length of the response or ERROR. |
| ERRNO | S_resolvLib_TRY_AGAIN ECONNREFUSE ETIMEDOU |

SEE ALSO *resolvLib, resolvGetHostByName(), resolvGetHostByAddr(), resolvDNExpand(), resolvDNComp(), resolvInit(), resolvParamsSet(), resolvParamsGet(), resolvMkQuery(), resolvQuery()*

rewind()

NAME *rewind()* – set the file position indicator to the beginning of a file (ANSI)

SYNOPSIS

```
void rewind
(
    FILE * fp /* stream */
)
```

DESCRIPTION This routine sets the file position indicator for a specified stream to the beginning of the file.

It is equivalent to:

```
(void) fseek (fp, 0L, SEEK_SET);
```

except that the error indicator for the stream is cleared.

INCLUDE FILES *stdio.h*

RETURNS N/A

SEE ALSO *ansiStdio, fseek(), ftell()*

rewinddir()

NAME *rewinddir()* – reset position to the start of a directory (POSIX)

SYNOPSIS

```
void rewinddir
(
    DIR * pDir /* pointer to directory descriptor */
)
```

DESCRIPTION This routine resets the position pointer in a directory descriptor (DIR). The *pDir* parameter is the directory descriptor pointer that was returned by *opendir()*.

As a result, the next *readdir()* will cause the current directory data to be read in again, as if an *opendir()* had just been performed. Any changes in the directory that have occurred since the initial *opendir()* will now be visible. The first entry in the directory will be returned by the next *readdir()*.

RETURNS N/A

SEE ALSO *dirLib*, *opendir()*, *readdir()*, *closedir()*

rindex()

NAME *rindex()* – find the last occurrence of a character in a string

SYNOPSIS

```
char *rindex
(
    const char * s, /* string in which to find character */
    int      c /* character to find in string */
)
```

DESCRIPTION This routine finds the last occurrence of character *c* in string *s*.

RETURNS A pointer to *c*, or NULL if *c* is not found.

SEE ALSO *bLib*

rip()

NAME *rip()* – return the contents of register **rip** (i960)

SYNOPSIS

```
int rip
(
    int taskId /* task ID, 0 means default task */
)
```

DESCRIPTION This command extracts the contents of register **rip**, the return instruction pointer, from the TCB of a specified task. If *taskId* is omitted or 0, the current default task is assumed.

RETURNS The contents of the **rip** register.

SEE ALSO *dbgArchLib, VxWorks Programmer's Guide: Target Shell*

ripAuthHook()

NAME *ripAuthHook()* – sample authentication hook

SYNOPSIS

```
STATUS ripAuthHook
(
    char *    pKey, /* rip2IfConfAuthKey entry from MIB-II family */
    RIP_PKT * pRip /* received RIP message */
)
```

DESCRIPTION This hook demonstrates one possible authentication mechanism. It rejects all RIP-2 messages which used simple password authentication since they did not match the key contained in the MIB variable. All other RIP-2 messages are also rejected since no other authentication type is supported and all RIP-1 messages are also rejected, as recommended by the RFC specification. This behavior is the same as if no hook were installed.

RETURNS OK if message is acceptable, or ERROR otherwise.

ERRNO N/A

SEE ALSO *ripLib*

ripAuthHookAdd()

NAME *ripAuthHookAdd()* – add an authentication hook to a RIP interface

SYNOPSIS

```
STATUS ripAuthHookAdd
(
    char*    pIpAddr, /* IP address in dotted decimal notation */
    FUNCPTR pAuthHook /* routine to handle message authentication */
)
```

DESCRIPTION This routine installs a hook routine to validate incoming RIP messages for a registered interface given by *pIpAddr*. (Interfaces created or changed after a RIP session has started may be installed/updated with the *ripIfSearch()* and *ripIfReset()* routines). The hook is

only called if an SNMP agent enables authentication for the corresponding interface. It uses the following prototype:

```
STATUS ripAuthHookRtn (char *pKey, RIP_PKT *pRip);
```

The first argument contains the authentication key for the message stored in the rip2IfConfAuthKey MIB variable and the second argument uses the RIP_PKT structure (defined in **rip/ripLib.h**) to access the message body. The routine must return OK if the message is acceptable, or ERROR otherwise. All RIP-2 messages sent to that routine already contain an authentication entry, but have not been verified. (Any unauthenticated RIP-2 messages have already been discarded as required by the RFC specification). RIP-1 messages may be accepted or rejected. RIP-2 messages requesting simple password authentication which match the key are accepted automatically before the hook is called. The remaining RIP-2 messages either did not match that key or are using an unknown authentication type. If any messages are rejected, the MIB-II counters are updated appropriately outside of the hook routine.

The current RIP implementation contains a sample authentication hook which may be added as follows:

```
if (ripAuthHookAdd ("90.0.0.1", ripAuthHook) == ERROR)
    logMsg ("Unable to add authorization hook.\n", 0, 0, 0, 0, 0, 0);
```

The sample routine only supports simple password authentication against the key included in the MIB variable. Since all such messages have already been accepted, all RIP-2 messages received by the routine are discarded. All RIP-1 messages are also discarded, so the hook actually has no effect. The body of that routine is:

```
STATUS ripAuthHook
(
    char *    pKey,    /* rip2IfConfAuthKey entry from MIB-II family */
    RIP_PKT * pRip    /* received RIP message */
)
{
    if (pRip->rip_vers == 1)
    {
        /*
         * @ The RFC specification recommends, but does not require, rejecting
         * @ version 1 packets when authentication is enabled.
         */
        return (ERROR);
    }
    /*
     * @ The authentication type field in the RIP message corresponds to
     * @ the first two bytes of the sa_data field overlaid on that
     * @ message by the sockaddr structure contained within the RIP_PKT
     * @ structure (see rip/ripLib.h).
     */
    if ( (pRip->rip_nets[0].rip_dst.sa_data[0] != 0) ||
```

```

        (pRip->rip_nets[0].rip_dst.sa_data[1] !=
M2_rip2IfConfAuthType_simplePassword)
    {
        /* Unrecognized authentication type. */
        return (ERROR);
    }
/*
 @ Discard version 2 packets requesting simple password authentication
 @ which did not match the MIB variable.
 */
return (ERROR);
}

```

A comparison against a different key could be performed as follows:

```

bzero ( (char *)&key, AUTHKEYLEN); /* AUTHKEYLEN from rip/m2RipLib.h */
/*
 @ The start of the authorization key corresponds to the third byte
 @ of the sa_data field in the sockaddr structure overlayed on the
 @ body of the RIP message by the RIP_PKT structure. It continues
 @ for the final 14 bytes of that structure and the first two bytes
 @ of the following rip_metric field.
 */
bcopy ( (char *) (pRip->rip_nets[0].rip_dst.sa_data + 2),
        (char *)&key, AUTHKEYLEN);
if (bcmp ( (char *)key, privateKey, AUTHKEYLEN) != 0)
    {
        /* Key does not match: reject message. */
        return (ERROR);
    }
return (OK);

```

The *ripAuthHookDelete()* routine will remove the installed function. If authentication is still enabled for the interface, all incoming messages which do not use simple password authentication will be rejected until a routine is provided.

| | |
|-----------------|--|
| RETURNS | OK if hook added, or ERROR otherwise. |
| ERRNO | S_m2Lib_INVALID_PARAMETER S_m2Lib_ENTRY_NOT_FOUND |
| SEE ALSO | ripLib |

ripAuthHookDelete()

| | |
|--------------------|---|
| NAME | <i>ripAuthHookDelete()</i> – remove an authentication hook from a RIP interface |
| SYNOPSIS | <pre>STATUS ripAuthHookDelete (char* pIpAddr /* IP address in dotted decimal notation */)</pre> |
| DESCRIPTION | This routine removes an assigned authentication hook from a registered interface indicated by <i>pIpAddr</i> . (Interfaces created or changed after a RIP session has started may be installed/updated with the <i>ripIfSearch()</i> and <i>ripIfReset()</i> routines). If authentication is still enabled for the interface, RIP-2 messages using simple password authentication will be accepted if they match the key in the MIB variable, but all other incoming messages will be rejected until a routine is provided. |
| RETURNS | OK, or ERROR if the interface could not be found. |
| ERRNO | S_m2Lib_INVALID_PARAMETER S_m2Lib_ENTRY_NOT_FOUND |
| SEE ALSO | <i>ripLib</i> |

ripDebugLevelSet()

| | |
|--------------------|--|
| NAME | <i>ripDebugLevelSet()</i> – specify amount of debugging output |
| SYNOPSIS | <pre>void ripDebugLevelSet (int level /* verbosity level (0 - 3) */)</pre> |
| DESCRIPTION | This routine influences the amount of debugging information sent to standard output during the RIP session. Higher values of the <i>level</i> parameter result in increasingly verbose output. A <i>level</i> of zero restores the default behavior by disabling all debugging output. |
| RETURNS | N/A |
| ERRNO | N/A |
| SEE ALSO | <i>ripLib</i> |

ripFilterDisable()

| | |
|--------------------|--|
| NAME | <i>ripFilterDisable()</i> – prevent strict border gateway filtering |
| SYNOPSIS | <code>void ripFilterDisable (void)</code> |
| DESCRIPTION | <p>This routine configures an active RIP session to ignore the restrictions necessary for RIP-1 and RIP-2 routers to operate correctly in the same network. All border gateway filtering is ignored and all routes to subnets, supernets, and specific hosts will be sent over any available interface. This operation is only correct if no RIP-1 routers are present anywhere on the network. Results are unpredictable if that condition is not met, but high rates of packet loss and widespread routing failures are likely.</p> <p>The border gateway filtering rules are in force by default.</p> |
| RETURNS | N/A |
| ERRNO | N/A |
| SEE ALSO | ripLib |

ripFilterEnable()

| | |
|--------------------|---|
| NAME | <i>ripFilterEnable()</i> – activate strict border gateway filtering |
| SYNOPSIS | <code>void ripFilterEnable (void)</code> |
| DESCRIPTION | <p>This routine configures an active RIP session to enforce the restrictions necessary for RIP-1 and RIP-2 routers to operate correctly in the same network as described in section 3.2 of RFC 1058 and section 3.3 of RFC 1723. When enabled, routes to portions of a logical network (including host routes) will be limited to routers within that network. Updates sent outside that network will only include a single entry representing the entire network. That entry will subsume all subnets and host-specific routes. If supernets are used, the entry will advertise the largest class-based portion of the supernet reachable through the connected interface.</p> |
| RETURNS | N/A |
| ERRNO | N/A |
| SEE ALSO | ripLib |

ripIfReset()

| | |
|--------------------|--|
| NAME | <i>ripIfReset()</i> – alter the RIP configuration after an interface changes |
| SYNOPSIS | <pre>STATUS ripIfReset (char * pIfName /* name of changed interface */)</pre> |
| DESCRIPTION | This routine updates the interface list and routing tables to reflect address and/or netmask changes for the device indicated by <i>pIfName</i> . To accommodate possible changes in the network number, all routes using the named interface are removed from the routing tables, but will be added in the next route update if appropriate. None of the removed routes are poisoned, so it may take some time for the routing tables of all the RIP participants to stabilize if the network number has changed. |
| RETURNS | OK, or ERROR if named interface not found or not added to list. |
| ERRNO | N/A |
| SEE ALSO | <i>ripLib</i> |

ripIfSearch()

| | |
|--------------------|---|
| NAME | <i>ripIfSearch()</i> – add new interfaces to the internal list |
| SYNOPSIS | <pre>void ripIfSearch (void)</pre> |
| DESCRIPTION | By default, a RIP session will not recognize any interfaces initialized after it has started. This routine schedules a search for additional interfaces which will occur during the next update of the internal routing table. Once completed, the session will accept and send RIP messages over the new interfaces. |
| RETURNS | N/A |
| ERRNO | N/A |
| SEE ALSO | <i>ripLib</i> |

ripLeakHookAdd()

NAME *ripLeakHookAdd()* – add a hook to bypass the RIP and kernel routing tables

SYNOPSIS

```
STATUS ripLeakHookAdd
(
    char * pIpAddr, /* IP address in dotted decimal notation */
    FUNCPTR pLeakHook /* function pointer to hook */
)
```

DESCRIPTION This routine installs a hook routine to support alternative routing protocols for the registered interface given by *pIpAddr*. (Interfaces created or changed after a RIP session has started may be installed/updated with the *ripIfSearch()* and *ripIfReset()* routines).

The hook uses the following interface:

```
STATUS ripLeakHookRtn (long dest, long gateway, long netmask)
```

The RIP session will not add the given route to any tables if the hook routine returns OK, but will create a route entry otherwise.

The *ripLeakHookDelete()* will allow the RIP session to add new routes unconditionally.

RETURNS OK, or ERROR if the interface could not be found.

ERRNO S_m2Lib_INVALID_PARAMETER
S_m2Lib_ENTRY_NOT_FOUND

SEE ALSO ripLib

ripLeakHookDelete()

NAME *ripLeakHookDelete()* – remove a table bypass hook from a RIP interface

SYNOPSIS

```
STATUS ripLeakHookDelete
(
    char* pIpAddr /* IP address in dotted decimal notation */
)
```

DESCRIPTION This routine removes the assigned bypass hook from a registered interface indicated by *pIpAddr*. (Interfaces created or changed after a RIP session has started may be installed/updated with the *ripIfSearch()* and *ripIfReset()* routines). The RIP session will

return to the default behavior and add entries to the internal RIP table and kernel routing table unconditionally.

RETURNS OK, or ERROR if the interface could not be found.

ERRNO S_m2Lib_INVALID_PARAMETER
S_m2Lib_ENTRY_NOT_FOUND

SEE ALSO ripLib

ripLibInit()

NAME *ripLibInit()* – initialize the RIP routing library

SYNOPSIS

```
STATUS ripLibInit
(
    BOOL supplier,          /* operate in silent mode? */
    BOOL gateway,          /* act as gateway to the Internet? */
    BOOL multicast,        /* use multicast or broadcast addresses? */
    int  version,          /* 1 or 2: selects format of outgoing messages */
    int  timerRate,        /* update frequency for internal routing table */
    int  supplyInterval,   /* update frequency for neighboring routers */
    int  expire,           /* maximum interval for renewing learned routes */
    int  garbage           /* elapsed time before deleting stale route */
)
```

DESCRIPTION This routine creates and initializes the global data structures used by the RIP routing library and starts a RIP session to maintain routing tables for a host. It must be called before using any other library routines, and is invoked automatically if `INCLUDE_RIP` is defined at the time the system is built.

The resulting RIP session will monitor all network interfaces which are currently available for messages from other RIP routers. If the *supplier* parameter is true, it will also respond to specific requests from other routers and transmit route updates over every known interface at the interval specified by *supplyInterval*.

Specifying a *gateway* setting of true establishes this router as a gateway to the wider Internet, capable of routing packets anywhere within the local networks. The final *multicast* flag indicates whether the RIP messages are sent to the pre-defined multicast address of 224.0.0.9 (which requires a *version* setting of 2) or to the broadcast address of the interfaces.

The *version* parameter determines the format used for outgoing RIP messages, and also sets the initial settings of the MIB-II compatibility switches in combination with the

multicast flag. A *version* of 1 will restrict all incoming traffic to that older message type. A *version* of 2 will set the receive switch to accept either type unless *multicast* is true, which limits reception to version 2 messages only. SNMP agents may alter those settings on a per-interface basis once startup is complete.

The remaining parameters set various system timers used to maintain the routing table. All of the values are expressed in seconds, and must be greater than or equal to 1. The *timerRate* determines how often the routing table is examined for changes and expired routes. The *supplyInterval* must be an exact multiple of that value. The *expire* parameter specifies the maximum time between updates before a route is invalidated and removed from the kernel table. Expired routes are then deleted from the internal RIP routing table if no update has been received within the time set by the *garbage* parameter.

The defaults for all the parameter settings are given by the following constants. The default timer values match the settings indicated in the RFC specification.

| Parameter Name | Default Value | Symbolic Constant |
|-----------------------|---------------|---------------------|
| <i>supplier</i> | 0 (FALSE) | RIP_SUPPLIER |
| <i>gateway</i> | 0 (FALSE) | RIP_GATEWAY |
| <i>multicast</i> | 0 (FALSE) | RIP_EXPIRE_TIME |
| <i>version</i> | 1 | RIP_SUPPLY_INTERVAL |
| <i>timerRate</i> | 1 | RIP_TIMER_RATE |
| <i>supplyInterval</i> | 30 | RIP_SUPPLY_INTERVAL |
| <i>expire</i> | 180 | RIP_EXPIRE_TIME |
| <i>garbage</i> | 300 | RIP_GARBAGE_TIME |

RETURNS OK, or ERROR if configuration fails.

ERRNO N/A

SEE ALSO *ripLib*

ripRouteShow()

NAME *ripRouteShow()* – display the internal routing table maintained by RIP

SYNOPSIS `void ripRouteShow()`

DESCRIPTION This routine prints every entry in the local RIP routing table. The flags displayed below the destination, gateway, and netmask addresses indicate the current route status. Entries with the `RTS_INTERFACE` flag indicate routes to directly connected networks which are

ripSendHookAdd()

generated locally. If `RTS_SUBNET` is set for an entry, it is subject to border gateway filtering (if enabled). When `RTS_INTERNAL` is also present, the corresponding entry is an "artificial" route created to supply distant networks with legitimate destinations if border filtering excludes the actual entry. Those entries are not copied to the kernel routing table. The `RTS_CHANGED` flag marks entries added or modified in the last timer interval which will be included in a triggered update.

| | |
|-----------------|---------------|
| RETURNS | N/A |
| ERRNO | N/A |
| SEE ALSO | ripLib |

ripSendHookAdd()

NAME *ripSendHookAdd()* – add an update filter to a RIP interface

SYNOPSIS

```
STATUS ripSendHookAdd
(
    char* pIpAddr,      /* IP address in dotted decimal notation */
    BOOL (*ripSendHook) (struct rt_entry* pRt) /* Routine to use. */
)
```

DESCRIPTION This routine installs a hook routine to screen individual route entries for inclusion in a periodic update. The routine is installed for the registered interface given by *pIpAddr*. (Interfaces created or changed after a RIP session has started may be installed/updated with the *ripIfSearch()* and *ripIfReset()* routines).

The hook uses the following prototype:

```
BOOL ripSendHookRtn (struct rt_entry* pRt);
```

If the hook returns `FALSE`, the route is not included in the update. Otherwise, it is included if it meets the other restrictions, such as simple split horizon and border gateway filtering. The *ripSendHookDelete()* routine removes this additional filter from the output processing.

| | |
|-----------------|--|
| RETURNS | OK, or ERROR if the interface could not be found. |
| ERRNO | <code>S_m2Lib_INVALID_PARAMETER</code> <code>S_m2Lib_ENTRY_NOT_FOUND</code> |
| SEE ALSO | ripLib |

ripSendHookDelete()

| | |
|--------------------|---|
| NAME | <i>ripSendHookDelete()</i> – remove an update filter from a RIP interface |
| SYNOPSIS | <pre>STATUS ripSendHookDelete (char* pIpAddr /* IP address in dotted decimal notation */)</pre> |
| DESCRIPTION | This routine removes the hook routine that allowed additional screening of route entries in periodic updates from the registered interface indicated by <i>pIpAddr</i> . (Interfaces created or changed after a RIP session has started may be installed/updated with the <i>ripIfSearch()</i> and <i>ripIfReset()</i> routines). The RIP session will return to the default behavior and include any entries which meet the other restrictions (such as simple split horizon). |
| RETURNS | OK, or ERROR if the interface could not be found. |
| ERRNO | S_m2Lib_INVALID_PARAMETER S_m2Lib_ENTRY_NOT_FOUND |
| SEE ALSO | <i>ripLib</i> |

ripShutdown()

| | |
|--------------------|---|
| NAME | <i>ripShutdown()</i> – terminate all RIP processing |
| SYNOPSIS | <pre>STATUS ripShutdown (void)</pre> |
| DESCRIPTION | This routine "poisons" all routes in the current table by transmitting updates with an infinite metric for each entry over all available interfaces. It then halts all RIP processing and removes the associated tasks and data structures. When completed successfully, the RIP services are unavailable until restarted with the <i>ripLibInit()</i> routine. |
| RETURNS | OK if shutdown completed, or ERROR otherwise. |
| ERRNO | N/A |
| SEE ALSO | <i>ripLib</i> |

rlogin()

- NAME** *rlogin()* – log in to a remote host
- SYNOPSIS**
- ```
STATUS rlogin
(
 char * host /* name of host to connect to */
)
```
- DESCRIPTION** This routine allows users to log in to a remote host. It may be called from the VxWorks shell as follows:
- ```
-> rlogin "remoteSystem"
```
- where *remoteSystem* is either a host name, which has been previously added to the remote host table by a call to *hostAdd()*, or an Internet address in dot notation (e.g., "90.0.0.2"). The remote system will be logged into with the current user name as set by a call to *iam()*. The user disconnects from the remote system by typing:
- ```
~.
```
- as the only characters on the line, or by simply logging out from the remote system using *logout()*.
- RETURNS** OK, or ERROR if the host is unknown, no privileged ports are available, the routine is unable to connect to the host, or the child process cannot be spawned.
- SEE ALSO** *rlogLib*, *iam()*, *logout()*

---

## *rlogind()*

- NAME** *rlogind()* – the VxWorks remote login daemon
- SYNOPSIS**
- ```
void rlogind (void)
```
- DESCRIPTION** This routine provides a facility for remote users to log in to VxWorks over the network. If the configuration macro `INCLUDE_RLOGIN` is defined, *rlogind()* is spawned by *rlogInit()* at boot time.
- Remote login requests will cause **stdin**, **stdout**, and **stderr** to be directed away from the console. When the remote user disconnects, **stdin**, **stdout**, and **stderr** are restored, and the shell is restarted. The *rlogind()* routine uses the remote user verification protocol specified by the UNIX remote shell daemon documentation, but ignores all the

information except the user name, which is used to set the VxWorks remote identity (see the manual entry for *iam()*).

The remote login daemon requires the existence of a pseudo-terminal device, which is created by *rlogInit()* before *rlogind()* is spawned. The *rlogind()* routine creates two child processes, *tRlogInTask* and *tRlogOutTask*, whenever a remote user is logged in. These processes exit when the remote connection is terminated.

RETURNS N/A

SEE ALSO *rlogLib*, *rlogInit()*, *iam()*

rlogInit()

NAME *rlogInit()* – initialize the remote login facility

SYNOPSIS `STATUS rlogInit (void)`

DESCRIPTION This routine initializes the remote login facility. It creates a pty (pseudo tty) device and spawns *rlogind()*. If the configuration macro `INCLUDE_RLOGIN` is defined, *rlogInit()* is called automatically at boot time.

RETURNS OK or ERROR.

SEE ALSO *rlogLib*, *ptyDrv*

rm()

NAME *rm()* – remove a file

SYNOPSIS `STATUS rm`
`(`
`char * fileName /* name of file to remove */`
`)`

DESCRIPTION This command is provided for UNIX similarity. It simply calls *remove()*.

RETURNS OK, or ERROR if the file cannot be removed.

SEE ALSO *usrLib*, *remove()*, *VxWorks Programmer's Guide: Target Shell*

rmdir()

| | |
|--------------------|---|
| NAME | <i>rmdir()</i> – remove a directory |
| SYNOPSIS | <pre>STATUS rmdir (char * dirName /* name of directory to remove */)</pre> |
| DESCRIPTION | <p>This command removes an existing directory from a hierarchical file system. The <i>dirName</i> string specifies the name of the directory to be removed, and may be either a full or relative pathname.</p> <p>This call is supported by the VxWorks NFS and dosFs file systems.</p> |
| RETURNS | OK, or ERROR if the directory cannot be removed. |
| SEE ALSO | <i>usrLib</i> , <i>mkdir()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> |

rngBufGet()

| | |
|--------------------|---|
| NAME | <i>rngBufGet()</i> – get characters from a ring buffer |
| SYNOPSIS | <pre>int rngBufGet (RING_ID rngId, /* ring buffer to get data from */ char * buffer, /* pointer to buffer to receive data */ int maxbytes /* maximum number of bytes to get */)</pre> |
| DESCRIPTION | <p>This routine copies bytes from the ring buffer <i>rngId</i> into <i>buffer</i>. It copies as many bytes as are available in the ring, up to <i>maxbytes</i>. The bytes copied will be removed from the ring.</p> |
| RETURNS | The number of bytes actually received from the ring buffer; it may be zero if the ring buffer is empty at the time of the call. |
| SEE ALSO | <i>rngLib</i> |

rngBufPut()

NAME *rngBufPut()* – put bytes into a ring buffer

SYNOPSIS

```
int rngBufPut
(
    RING_ID rngId, /* ring buffer to put data into */
    char * buffer, /* buffer to get data from */
    int nbytes /* number of bytes to try to put */
)
```

DESCRIPTION This routine puts bytes from *buffer* into ring buffer *ringId*. The specified number of bytes will be put into the ring, up to the number of bytes available in the ring.

RETURNS The number of bytes actually put into the ring buffer; it may be less than number requested, even zero, if there is insufficient room in the ring buffer at the time of the call.

SEE ALSO *rngLib*

rngCreate()

NAME *rngCreate()* – create an empty ring buffer

SYNOPSIS

```
RING_ID rngCreate
(
    int nbytes /* number of bytes in ring buffer */
)
```

DESCRIPTION This routine creates a ring buffer of size *nbytes*, and initializes it. Memory for the buffer is allocated from the system memory partition.

RETURNS The ID of the ring buffer, or NULL if memory cannot be allocated.

SEE ALSO *rngLib*

rngDelete()

| | |
|--------------------|--|
| NAME | <i>rngDelete()</i> – delete a ring buffer |
| SYNOPSIS | <pre>void rngDelete (RING_ID ringId /* ring buffer to delete */)</pre> |
| DESCRIPTION | This routine deletes a specified ring buffer. Any data currently in the buffer will be lost. |
| RETURNS | N/A |
| SEE ALSO | rngLib |

rngFlush()

| | |
|--------------------|--|
| NAME | <i>rngFlush()</i> – make a ring buffer empty |
| SYNOPSIS | <pre>void rngFlush (RING_ID ringId /* ring buffer to initialize */)</pre> |
| DESCRIPTION | This routine initializes a specified ring buffer to be empty. Any data currently in the buffer will be lost. |
| RETURNS | N/A |
| SEE ALSO | rngLib |

rngFreeBytes()

NAME *rngFreeBytes()* – determine the number of free bytes in a ring buffer

SYNOPSIS

```
int rngFreeBytes
(
    RING_ID ringId /* ring buffer to examine */
)
```

DESCRIPTION This routine determines the number of bytes currently unused in a specified ring buffer.

RETURNS The number of unused bytes in the ring buffer.

SEE ALSO *rngLib*

rngIsEmpty()

NAME *rngIsEmpty()* – test if a ring buffer is empty

SYNOPSIS

```
BOOL rngIsEmpty
(
    RING_ID ringId /* ring buffer to test */
)
```

DESCRIPTION This routine determines if a specified ring buffer is empty.

RETURNS TRUE if empty, FALSE if not.

SEE ALSO *rngLib*

rngIsFull()

NAME *rngIsFull()* – test if a ring buffer is full (no more room)

SYNOPSIS

```
BOOL rngIsFull
(
    RING_ID ringId /* ring buffer to test */
)
```

DESCRIPTION This routine determines if a specified ring buffer is completely full.

RETURNS TRUE if full, FALSE if not.

SEE ALSO **rngLib**

rngMoveAhead()

NAME *rngMoveAhead()* – advance a ring pointer by *n* bytes

SYNOPSIS

```
void rngMoveAhead
(
    RING_ID ringId, /* ring buffer to be advanced */
    int     n       /* number of bytes ahead to move input pointer */
)
```

DESCRIPTION This routine advances the ring buffer input pointer by *n* bytes. This makes *n* bytes available in the ring buffer, after having been written ahead in the ring buffer with *rngPutAhead()*.

RETURNS N/A

SEE ALSO **rngLib**

rngNBytes()

NAME *rngNBytes()* – determine the number of bytes in a ring buffer

SYNOPSIS

```
int rngNBytes
(
    RING_ID ringId /* ring buffer to be enumerated */
)
```

DESCRIPTION This routine determines the number of bytes currently in a specified ring buffer.

RETURNS The number of bytes filled in the ring buffer.

SEE ALSO *rngLib*

rngPutAhead()

NAME *rngPutAhead()* – put a byte ahead in a ring buffer without moving ring pointers

SYNOPSIS

```
void rngPutAhead
(
    RING_ID ringId, /* ring buffer to put byte in */
    char    byte,   /* byte to be put in ring */
    int     offset  /* offset beyond next input byte where to put byte */
)
```

DESCRIPTION This routine writes a byte into the ring, but does not move the ring buffer pointers. Thus the byte will not yet be available to *rngBufGet()* calls. The byte is written *offset* bytes ahead of the next input location in the ring. Thus, an offset of 0 puts the byte in the same position as would *RNG_ELEM_PUT* would put a byte, except that the input pointer is not updated.

Bytes written ahead in the ring buffer with this routine can be made available all at once by subsequently moving the ring buffer pointers with the routine *rngMoveAhead()*.

Before calling *rngPutAhead()*, the caller must verify that at least *offset + 1* bytes are available in the ring buffer.

RETURNS N/A

SEE ALSO *rngLib*

romStart()

NAME *romStart()* – generic ROM initialization

SYNOPSIS

```
void romStart
(
    int startType /* start type */
)
```

DESCRIPTION This is the first C code executed after reset.

This routine is called by the assembly start-up code in *romInit()*. It clears memory, copies ROM to RAM, and possibly invokes the uncompressor. It then jumps to the entry point of the uncompressed object code.

RETURNS N/A

SEE ALSO *bootInit*

round()

NAME *round()* – round a number to the nearest integer

SYNOPSIS

```
double round
(
    double x /* value to round */
)
```

DESCRIPTION This routine rounds a double-precision value *x* to the nearest integral value.

INCLUDE FILES *math.h*

RETURNS The double-precision representation of *x* rounded to the nearest integral value.

SEE ALSO *mathALib*

roundf()

| | |
|----------------------|--|
| NAME | <i>roundf()</i> – round a number to the nearest integer |
| SYNOPSIS | <pre>float roundf (float x /* argument */)</pre> |
| DESCRIPTION | This routine rounds a single-precision value <i>x</i> to the nearest integral value. |
| INCLUDE FILES | <code>math.h</code> |
| RETURNS | The single-precision representation of <i>x</i> rounded to the nearest integral value. |
| SEE ALSO | <code>mathALib</code> |

routeAdd()

| | |
|--------------------|--|
| NAME | <i>routeAdd()</i> – add a route |
| SYNOPSIS | <pre>STATUS routeAdd (char * destination, /* inet addr or name of route destination */ char * gateway /* inet addr or name of gateway to destination */)</pre> |
| DESCRIPTION | <p>This routine adds gateways to the network routing tables. It is called from a VxWorks machine that needs to establish a gateway to a destination network (or machine).</p> <p>You can specify both <i>destination</i> and <i>gateway</i> in standard Internet address format (for example, 90.0.0.2), or you can specify them using their host names, as specified with <i>hostAdd()</i>.</p> |
| EXAMPLE | <p>Consider the following example:</p> <pre>-> routeAdd "90.0.0.0", "gate"</pre> <p>This call tells VxWorks that the machine with the host name "gate" is the gateway to network 90.0.0.0. The host "gate" must already have been created by <i>hostAdd()</i>.</p> <p>Consider the following example:</p> |

routeDelete()

```
-> routeAdd "90.0.0.0", "91.0.0.3"
```

This call tells VxWorks that the machine with the Internet address 91.0.0.3 is the gateway to network 90.0.0.0.

Consider the following example:

```
-> routeAdd "destination", "gate"
```

This call tells VxWorks that the machine with the host name "gate" is the gateway to the machine named "destination". The host names "gate" and "destination" must already have been created by *hostAdd()*.

Consider the following example:

```
-> routeAdd "0", "gate"
```

This call tells VxWorks that the machine with the host name "gate" is the default gateway. The host "gate" must already have been created by *hostAdd()*. A default gateway is where Internet Protocol (IP) datagrams are routed when there is no specific routing table entry available for the destination IP network or host.

RETURNS OK or ERROR.

SEE ALSO *routeLib*

routeDelete()

NAME *routeDelete()* – delete a route

SYNOPSIS

```
STATUS routeDelete
(
    char * destination, /* inet addr or name of route destination */
    char * gateway      /* inet addr or name of gateway to destination */
)
```

DESCRIPTION This routine deletes a specified route from the network routing tables.

RETURNS OK or ERROR.

SEE ALSO *routeLib*, *routeAdd()*

routeNetAdd()

| | |
|--------------------|--|
| NAME | <i>routeNetAdd()</i> – add a route to a destination that is a network |
| SYNOPSIS | <pre>STATUS routeNetAdd (char * destination, /* inet addr or name of network destination */ char * gateway /* inet addr or name of gateway to destination */)</pre> |
| DESCRIPTION | This routine is equivalent to <i>routeAdd()</i> , except that the destination address is assumed to be a network. This is useful for adding a route to a sub-network that is not on the same overall network as the local network. |
| RETURNS | OK or ERROR. |
| SEE ALSO | <i>routeLib</i> |

routeProtoPrioritySet()

| | |
|--------------------|--|
| NAME | <i>routeProtoPrioritySet()</i> – set the priority of routes added by the routing protocol |
| SYNOPSIS | <pre>STATUS routeProtoPrioritySet (int proto, /* protocol no, from m2Lib.h */ int prio /* priority, >= 0 , <= 200 */)</pre> |
| DESCRIPTION | <p>This routine assigns a priority to a routing protocol. A route generated by the <i>proto</i> protocol is added to the routing table only if a protocol of higher priority does not already have that route installed in the table. Use <i>proto</i> to identify the protocol. See m2Lib.h for a listing of valid values for <i>proto</i>. Use <i>prio</i> to specify the priority level you want to assign to <i>proto</i>. The <i>prio</i> parameter may be any integer value greater or equal to 0 and less than or equal to 200. The higher values indicate higher priority. If you do not want VxWorks to prioritize protocols, do not call this routine.</p> <p>Routes that are added with the <i>routeAdd()</i> or <i>mRouteAdd()</i> call are of type M2_ipRouteProto_other. These are static routes that are not affected by routing protocols such as RIP and OSPF. To change the priority of routes added in this way pass the value M2_ipRoute_Proto_other in the first argument of this routine.</p> |

routeShow()

RETURNS OK if priority set successfully else ERROR.

SEE ALSO **routeLib**

routeShow()

NAME *routeShow()* – display host and network routing tables

SYNOPSIS `void routeShow (void)`

DESCRIPTION This routine displays the current routing information contained in the routing table.

EXAMPLE

```

-> routeShow
ROUTE NET TABLE
destination      gateway      flags  Refcnt  Use      Interface
-----
90.0.0.0         90.0.0.63   1      1       142     enp0
-----
ROUTE HOST TABLE
destination      gateway      flags  Refcnt  Use      Interface
-----
127.0.0.1       127.0.0.1   5      0       82      lo0
-----

```

The flags field represents a decimal value of the flags specified for a given route. The following is a list of currently available flag values:

```

0x1      – route is usable (that is, "up")
0x2      – destination is a gateway
0x4      – host specific routing entry
0x8      – host or net unreachable
0x10     – created dynamically (by redirect)
0x20     – modified dynamically (by redirect)
0x40     – message confirmed
0x80     – subnet mask present
0x100    – generate new routes on use
0x200    – external daemon resolves name
0x400    – generated by ARP
0x800    – manually added (static)
0x1000   – just discard packets (during updates)
0x2000   – modified by management protocol
0x4000   – protocol specific routing flag
0x8000   – protocol specific routing flag

```

In the above display example, the entry in the ROUTE NET TABLE has a flag value of 1, which indicates that this route is "up" and usable and network specific (the 0x4 bit is turned off). The entry in the ROUTE HOST TABLE has a flag value of 5 (0x1 OR'ed with 0x4), which indicates that this route is "up" and usable and host-specific.

RETURNS N/A

SEE ALSO `netShow`

routeStatShow()

NAME `routeStatShow()` – display routing statistics

SYNOPSIS `void routeStatShow (void)`

DESCRIPTION This routine displays routing statistics.

RETURNS N/A

SEE ALSO `netShow`

rpcInit()

NAME `rpcInit()` – initialize the RPC package

SYNOPSIS `STATUS rpcInit (void)`

DESCRIPTION This routine must be called before any task can use the RPC facility; it spawns the portmap daemon. It is called automatically if the configuration macro `INCLUDE_RPC` is defined.

RETURNS OK, or ERROR if the portmap daemon cannot be spawned.

SEE ALSO `rpcLib`

rpcTaskInit()

| | |
|--------------------|---|
| NAME | <i>rpcTaskInit()</i> – initialize a task’s access to the RPC package |
| SYNOPSIS | <code>STATUS rpcTaskInit (void)</code> |
| DESCRIPTION | This routine must be called by a task before it makes any calls to other routines in the RPC package. |
| RETURNS | OK, or ERROR if there is insufficient memory or the routine is unable to add a task delete hook. |
| SEE ALSO | <code>rpcLib</code> |

rresvport()

| | |
|--------------------|--|
| NAME | <i>rresvport()</i> – open a socket with a privileged port bound to it |
| SYNOPSIS | <pre>int rresvport (int * alport /* port number to initially try */)</pre> |
| DESCRIPTION | This routine opens a socket with a privileged port bound to it. It is analogous to the UNIX routine <i>rresvport()</i> . |
| RETURNS | A socket descriptor, or ERROR if either the socket cannot be opened or all ports are in use. |
| SEE ALSO | <code>remLib</code> , UNIX BSD 4.3 manual entry for <i>rresvport()</i> |

rt11FsDateSet()

| | |
|-----------------|--|
| NAME | <i>rt11FsDateSet()</i> – set the rt11Fs file system date |
| SYNOPSIS | <pre>void rt11FsDateSet (int year, /* year (72...03 (RT-11's days are numbered)) */</pre> |

```

int month, /* month (0, or 1...12) */
int day   /* day (0, or 1...31) */
)

```

DESCRIPTION This routine sets the date for the rt11Fs file system, which remains in effect until changed. All files created are assigned this creation date.

To set a blank date, invoke the command:

```
rt11FsDateSet (72, 0, 0); /* a date outside RT-11's epoch */
```

NOTE No automatic incrementing of the date is performed; each new date must be set with a call to this routine.

RETURNS N/A

SEE ALSO *rt11FsLib*

rt11FsDevInit()

NAME *rt11FsDevInit()* – initialize the rt11Fs device descriptor

SYNOPSIS

```

RT_VOL_DESC *rt11FsDevInit
(
    char *    devName,      /* device name */
    BLK_DEV * pBlkDev,     /* pointer to block device info */
    BOOL      rt11Fmt,     /* TRUE if RT-11 skew & interleave */
    int       nEntries,    /* no. of dir entries incl term entry */
    BOOL      changeNoWarn /* TRUE if no disk change warning */
)

```

DESCRIPTION This routine initializes the device descriptor. The *pBlkDev* parameter is a pointer to an already-created **BLK_DEV** device structure. This structure contains definitions for various aspects of the physical device format, as well as pointers to the sector read, sector write, *ioctl()*, status check, and reset functions for the device.

The *rt11Fmt* parameter is TRUE if the device is to be accessed using standard RT-11 skew and interleave.

The device directory will consist of one segment able to contain at least as many files as specified by *nEntries*. If *nEntries* is equal to **RT_FILES_FOR_2_BLOCK_SEG**, strict RT-11 compatibility is maintained.

The *changeNoWarn* parameter is TRUE if the disk may be changed without announcing the change via *rt11FsReadyChange()*. Setting *changeNoWarn* to TRUE causes the disk to be

rt11FsInit()

regularly remounted, in case it has been changed. This results in a significant performance penalty.

- NOTE** An ERROR is returned if *rt11Fmt* is TRUE and the **bd_blkPerTrack**(sectors per track) field in the **BLK_DEV** structure is odd. This is because an odd number of sectors per track is incompatible with the RT-11 interleaving algorithm.
- RETURNS** A pointer to the volume descriptor (**RT_VOL_DESC**), or NULL if invalid device parameters were specified, or the routine runs out of memory.
- SEE ALSO** **rt11FsLib**

rt11FsInit()

- NAME** *rt11FsInit()* – prepare to use the rt11Fs library
- SYNOPSIS**
- ```
STATUS rt11FsInit
(
 int maxFiles /* max no. of simultaneously open rt11Fs files */
)
```
- DESCRIPTION** This routine initializes the rt11Fs library. It must be called exactly once, before any other routine in the library. The *maxFiles* parameter specifies the number of rt11Fs files that may be open at once. This routine initializes the necessary memory structures and semaphores.
- This routine is called automatically from the root task, *usrRoot()*, in **usrConfig.c** when the configuration macro **INCLUDE\_RT11FS** is defined.
- RETURNS** OK, or ERROR if memory is insufficient.
- SEE ALSO** **rt11FsLib**



---

## *rt11FsMkfs()*

- NAME** *rt11FsMkfs()* – initialize a device and create an rt11Fs file system
- SYNOPSIS**
- ```
RT_VOL_DESC *rt11FsMkfs
(
    char *    volName, /* volume name to use */
    BLK_DEV * pBlkDev /* pointer to block device struct */
)
```
- DESCRIPTION**
- This routine provides a quick method of creating an rt11Fs file system on a device. It is used instead of the two-step procedure of calling *rt11FsDevInit()* followed by an *ioctl()* call with an FIODISKINIT function code.
- This routine provides defaults for the rt11Fs parameters expected by *rt11FsDevInit()*. The directory size is set to RT_FILES_FOR_2_BLOCK_SEG(defined in *rt11FsLib.h*). No standard disk format is assumed; this allows the use of rt11Fs on block devices with an odd number of sectors per track. The *changeNoWarn* parameter is defined as FALSE, indicating that the disk will not be replaced without *rt11FsReadyChange()* being called first.
- If different values are needed for any of these parameters, the routine *rt11FsDevInit()* must be used instead of this routine, followed by a request for disk initialization using the *ioctl()* function FIODISKINIT.
- RETURNS** A pointer to an rt11Fs volume descriptor (RT_VOL_DESC), or NULL if there is an error.
- SEE ALSO** *rt11FsLib*, *rt11FsDevInit()*

rt11FsModeChange()

- NAME** *rt11FsModeChange()* – modify the mode of an rt11Fs volume
- SYNOPSIS**
- ```
void rt11FsModeChange
(
 RT_VOL_DESC * vdptr, /* pointer to volume descriptor */
 int newMode /* O_RDONLY, O_WRONLY, or O_RDWR (both) */
)
```

***rt11FsReadyChange()***

**DESCRIPTION** This routine sets the volume descriptor mode to *newMode*. It should be called whenever the read and write capabilities are determined, usually after a ready change. See the manual entry for *rt11FsReadyChange()*.

The *rt11FsDevInit()* routine initially sets the mode to **O\_RDWR**, (e.g., both **O\_RDONLY** and **O\_WRONLY**).

**RETURNS** N/A

**SEE ALSO** *rt11FsLib*, *rt11FsDevInit()*, *rt11FsReadyChange()*

***rt11FsReadyChange()***

**NAME** *rt11FsReadyChange()* – notify rt11Fs of a change in ready status

**SYNOPSIS**

```
void rt11FsReadyChange
(
 RT_VOL_DESC * vdptr /* pointer to device descriptor */
)
```

**DESCRIPTION** This routine sets the volume descriptor state to **RT\_VD\_READY\_CHANGED**. It should be called whenever a driver senses that a device has come on-line or gone off-line (e.g., a disk has been inserted or removed).

**RETURNS** N/A

**SEE ALSO** *rt11FsLib*

***s()***

**NAME** *s()* – single-step a task

**SYNOPSIS**

```
STATUS s
(
 int taskNameOrId, /* task to step; 0 = use default */
 INSTR * addr, /* address to step to; 0 = next instruction */
 INSTR * addr1 /* address for npc, 0 = next instruction */
)
```

- DESCRIPTION** This routine single-steps a task that is stopped at a breakpoint.  
 To execute, enter:
- ```
-> s [task[,addr[,addr1]]]
```
- If *task* is omitted or zero, the last task referenced is assumed. If *addr* is non-zero, then the program counter is changed to *addr*; if *addr1* is non-zero, the next program counter is changed to *addr1*, and the task is stepped.
- CAVEAT** When a task is continued, *s()* does not distinguish between a suspended task or a task suspended by the debugger. Therefore, its use should be restricted to only those tasks being debugged.
- NOTE** The next program counter, *addr1*, is currently supported only by SPARC.
- RETURNS** OK, or ERROR if the debugging package is not installed, the task cannot be found, or the task is not suspended.
- SEE ALSO** **dbgLib**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

sa1100DevInit()

- NAME** *sa1100DevInit()* – initialise an SA1100 channel
- SYNOPSIS**
- ```
void sa1100DevInit
(
 SA1100_CHAN * pChan /* ptr to SA1100_CHAN describing this channel */
)
```
- DESCRIPTION** This routine initialises some **SIO\_CHAN** function pointers and then resets the chip to a quiescent state. Before this routine is called, the BSP must already have initialised all the device addresses, etc. in the **SA1100\_CHAN** structure.
- RETURNS** N/A
- SEE ALSO** **sa1100Sio**

---

## **sa1100Int()**

**NAME** *sa1100Int()* – handle an interrupt

**SYNOPSIS**

```
void sa1100Int
(
 SA1100_CHAN * pChan /* ptr to SA1100_CHAN describing this channel */
)
```

**DESCRIPTION** This routine handles interrupts from the UART.

**RETURNS** N/A

**SEE ALSO** *sa1100Sio*

---

## **saIoWrite()**

**NAME** *saIoWrite()* – send a packet to the master agent’s message queue

**SYNOPSIS**

```
STATUS saIoWrite
(
 PTR_T ipchandle, /* Subagent’s identifier */
 EBUFFER_T * pBuf, /* Encoded buffer */
 INT_32_T code /* Message type */
)
```

**DESCRIPTION** This routine is called either from *snmpSaHandlerAsync()* or from the registration routines. *ipchandle* contains an identifier to the sub agents’s message queue except for the case when the message is a response to *IPC\_AYT*. In this case, it contains the identifier to the local queue at the master agent. The *pBuf* parameter points to the message being sent. The *code* parameter takes a value that indicates how the master agent should process the message. Value *code* values are *CALL\_QUERY\_HANDLER*, *CALL\_REG\_HANDLER*, and *IPC\_AYT*. For more on how these values influence message processing in the master agent, see the description of *snmpMonitorSpawn()*.

**RETURNS** OK or ERROR.

**SEE ALSO** *saIoLib*

---

## *saIpcFree()*

|                    |                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>saIpcFree()</i> – free the specified IPC mechanism                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>void saIpcFree (     PTR_T ipchandle /* pointer to IPC handle */ )</pre>                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | Call this routine to free the IPC mechanism specified by <i>ipchandle</i> . You created this IPC mechanism with a call to <i>snmpSaInit()</i> . If you rewrote <i>snmpSaInit()</i> to use an IPC mechanism other than message queues, you must rewrite <i>saIpcFree()</i> to match. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <i>saIoLib</i>                                                                                                                                                                                                                                                                      |

---

## *saMsgBuild()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>saMsgBuild()</i> – build and encode a message and send it to the master agent                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>void saMsgBuild (     VBL_T *      vblast, /* pointer to varbind list */     SA_HEADER_T * hdr_blk, /* pointer to header block */     SA_DEMUX_T * demuxer, /* pointer to demuxer */     PTR_T      saId /* IPC handle */ )</pre>                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b> | <p>The <i>hdrBlkCreate()</i> routine calls <i>saMsgBuild()</i> to build a message, encode it, and transmit it to the master agent. The message is built up from the information provided in the input parameters:</p> <p><i>vblast</i><br/>Expects a pointer to the <b>VBL_T</b> structure containing the varbind list you want to include in the message.</p> <p><i>hdr_blk</i><br/>Expects a pointer to the header for this message.</p> <p><i>demuxer</i><br/>Expects a pointer to the demuxer information for this message. The demuxer</p> |

**scanf()**

information consists of a string and an object ID. In a message dealing with a version 1 request, the string is the community string and the object ID is unused. In a message dealing with a version 2 request, the string is the local entity string from the context and the object ID is the local time object ID from the context.

*said*

Expects a pointer to the IPC mechanism (a message queue ID) that the master agent can use to respond to this message.

To encode the message, this routine calls *snmpSubEncode()*. To send the message to the master agent, this routine calls *saIoWrite()*.

**RETURNS** N/A

**SEE ALSO** **saIoLib**

**scanf()**

**NAME** *scanf()* – read and convert characters from the standard input stream (ANSI)

**SYNOPSIS**

```
int scanf
(
 char const * fmt /* format string */
)
```

**DESCRIPTION** This routine reads input from the standard input stream under the control of the string *fmt*. It is equivalent to *fscanf()* with an *fp* argument of **stdin**.

**INCLUDE FILES** **stdio.h**

**RETURNS** The number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure; or EOF if an input failure occurs before any conversion.

**SEE ALSO** **ansiStdio**, *fscanf()*, *sscanf()*

---

## *sched\_get\_priority\_max()*

|                    |                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>sched_get_priority_max()</i> – get the maximum priority (POSIX)                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>int sched_get_priority_max (     int policy /* scheduling policy */ )</pre>                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | This routine returns the value of the highest possible task priority for a specified scheduling policy ( <b>SCHED_FIFO</b> or <b>SCHED_RR</b> ).                                                                                                                                                                               |
| <b>NOTE</b>        | If the global variable <b>posixPriorityNumbering</b> is <b>FALSE</b> , the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers. |
| <b>RETURNS</b>     | Maximum priority value, or -1 (ERROR) on error.                                                                                                                                                                                                                                                                                |
| <b>ERRNO</b>       | <b>EINVAL</b><br>– invalid scheduling policy.                                                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>schedPxBLib</b>                                                                                                                                                                                                                                                                                                             |

---

## *sched\_get\_priority\_min()*

|                    |                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>sched_get_priority_min()</i> – get the minimum priority (POSIX)                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>int sched_get_priority_min (     int policy /* scheduling policy */ )</pre>                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | This routine returns the value of the lowest possible task priority for a specified scheduling policy ( <b>SCHED_FIFO</b> or <b>SCHED_RR</b> ).                                                                                                                                                                                |
| <b>NOTE</b>        | If the global variable <b>posixPriorityNumbering</b> is <b>FALSE</b> , the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers. |

**RETURNS** Minimum priority value, or -1 (ERROR) on error.

**ERRNO** EINVAL  
– invalid scheduling policy.

**SEE ALSO** schedPxBLib

---

## ***sched\_getparam()***

**NAME** *sched\_getparam()* – get the scheduling parameters for a specified task (POSIX)

**SYNOPSIS**

```
int sched_getparam
(
 pid_t tid, /* task ID */
 struct sched_param * param /* scheduling param to store priority */
)
```

**DESCRIPTION** This routine gets the scheduling priority for a specified task, *tid*. If *tid* is 0, it gets the priority of the calling task. The task's priority is copied to the **sched\_param** structure pointed to by *param*.

**NOTE** If the global variable **posixPriorityNumbering** is FALSE, the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers.

**RETURNS** 0 (OK) if successful, or -1 (ERROR) on error.

**ERRNO** ESRCH  
– invalid task ID.

**SEE ALSO** schedPxBLib



---

## *sched\_getscheduler()*

**NAME** *sched\_getscheduler()* – get the current scheduling policy (POSIX)

**SYNOPSIS**

```
int sched_getscheduler
(
 pid_t tid /* task ID */
)
```

**DESCRIPTION** This routine returns the current scheduling policy (i.e., SCHED\_FIFO or SCHED\_RR).

**RETURNS** Current scheduling policy (SCHED\_FIFO or SCHED\_RR), or -1 (ERROR) on error.

**ERRNO** ESRCH  
– invalid task ID.

**SEE ALSO** schedPxBLib

---

## *sched\_rr\_get\_interval()*

**NAME** *sched\_rr\_get\_interval()* – get the current time slice (POSIX)

**SYNOPSIS**

```
int sched_rr_get_interval
(
 pid_t tid, /* task ID */
 struct timespec * interval /* struct to store time slice */
)
```

**DESCRIPTION** This routine sets *interval* to the current time slice period if round-robin scheduling is currently enabled.

**RETURNS** 0 (OK) if successful, -1 (ERROR) on error.

**ERRNO** EINVAL  
– round-robin scheduling is not currently enabled.  
ESRCH  
– invalid task ID.

**SEE ALSO** schedPxBLib

---

## ***sched\_setparam()***

**NAME** *sched\_setparam()* – set a task’s priority (POSIX)

**SYNOPSIS**

```
int sched_setparam
(
 pid_t tid, /* task ID */
 const struct sched_param * param /* scheduling parameter */
)
```

**DESCRIPTION** This routine sets the priority of a specified task, *tid*. If *tid* is 0, it sets the priority of the calling task. Valid priority numbers are 0 through 255.

The *param* argument is a structure whose member **sched\_priority** is the integer priority value. For example, the following program fragment sets the calling task’s priority to 13 using POSIX interfaces:

```
#include "sched.h"
...
struct sched_param AppSchedPrio;
...
AppSchedPrio.sched_priority = 13;
if (sched_setparam (0, &AppSchedPrio) != OK)
{
 ... /* recovery attempt or abort message */
}
...
```

**NOTE** If the global variable **posixPriorityNumbering** is FALSE, the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers.

**RETURNS** 0 (OK) if successful, or -1 (ERROR) on error.

**ERRNO**

- EINVAL**  
– scheduling priority is outside valid range.
- ESRCH**  
– task ID is invalid.

**SEE ALSO** **schedPxLib**

---

## *sched\_setscheduler()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>sched_setscheduler()</i> – set scheduling policy and scheduling parameters (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>int sched_setscheduler (     pid_t          tid,    /* task ID */     int            policy, /* scheduling policy requested */     const struct sched_param * param /* scheduling parameters requested */ )</pre>                                                                                                                                                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | <p>This routine sets the scheduling policy and scheduling parameters for a specified task, <i>tid</i>. If <i>tid</i> is 0, it sets the scheduling policy and scheduling parameters for the calling task.</p> <p>Because VxWorks does not set scheduling policies (e.g., round-robin scheduling) on a task-by-task basis, setting a scheduling policy that conflicts with the current system policy simply fails and <i>errno</i> is set to <b>EINVAL</b>. If the requested scheduling policy is the same as the current system policy, then this routine acts just like <i>sched_setparam()</i>.</p> |
| <b>NOTE</b>        | <p>If the global variable <b>posixPriorityNumbering</b> is <b>FALSE</b>, the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers.</p>                                                                                                                                                                                                                                                                 |
| <b>RETURNS</b>     | The previous scheduling policy ( <b>SCHED_FIFO</b> or <b>SCHED_RR</b> ), or -1 ( <b>ERROR</b> ) on error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>ERRNO</b>       | <b>EINVAL</b><br>– scheduling priority is outside valid range, or it is impossible to set the specified scheduling policy.<br><b>ESRCH</b><br>– invalid task ID.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <i>schedPxBLib</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

---

## *sched\_yield()*

|                    |                                                          |
|--------------------|----------------------------------------------------------|
| <b>NAME</b>        | <i>sched_yield()</i> – relinquish the CPU (POSIX)        |
| <b>SYNOPSIS</b>    | <pre>int sched_yield (void)</pre>                        |
| <b>DESCRIPTION</b> | This routine forces the running task to give up the CPU. |

***scsi2IfInit()***

**RETURNS** 0 (OK) if successful, or -1 (ERROR) on error.

**SEE ALSO** **schedPxLib**

***scsi2IfInit()***

**NAME** *scsi2IfInit()* – initialize the SCSI-2 interface to scsiLib

**SYNOPSIS** `void scsi2IfInit ()`

**DESCRIPTION** This routine initializes the SCSI-2 function interface by adding all the routines in **scsi2Lib** plus those in **scsiDirectLib** and **scsiCommonLib**. It is invoked by **usrConfig.c** if the macro **INCLUDE\_SCSI2** is defined in **config.h**. The calling interface remains the same between SCSI-1 and SCSI-2; this routine simply sets the calling interface function pointers to the SCSI-2 functions.

**RETURNS** N/A

**SEE ALSO** **scsi2Lib**

***scsiAutoConfig()***

**NAME** *scsiAutoConfig()* – configure all devices connected to a SCSI controller

**SYNOPSIS** `STATUS scsiAutoConfig`  
`(`  
`SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */`  
`)`

**DESCRIPTION** This routine cycles through all valid SCSI bus IDs and logical unit numbers (LUNs), attempting a *scsiPhysDevCreate()* with default parameters on each. All devices which support the INQUIRY command are configured. The *scsiShow()* routine can be used to find the system table of SCSI physical devices attached to a specified SCSI controller. In addition, *scsiPhysDevIdGet()* can be used programmatically to get a pointer to the SCSI\_PHYS\_DEV structure associated with the device at a specified SCSI bus ID and LUN.

**RETURNS** OK, or ERROR if *pScsiCtrl* and the global variable **pSysScsiCtrl** are both NULL.

**SEE ALSO** **scsiLib**

---

## *scsiBlkDevCreate()*

- NAME** *scsiBlkDevCreate()* – define a logical partition on a SCSI block device
- SYNOPSIS**
- ```
BLK_DEV * scsiBlkDevCreate
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device info */
    int             numBlocks,    /* number of blocks in block device */
    int             blockOffset  /* address of first block in volume */
)
```
- DESCRIPTION** This routine creates and initializes a **BLK_DEV** structure, which describes a logical partition on a SCSI physical-block device. A logical partition is an array of contiguously addressed blocks; it can be completely described by the number of blocks and the address of the first block in the partition. In normal configurations partitions do not overlap, although such a condition is not an error.
- NOTE** If *numBlocks* is 0, the rest of device is used.
- RETURNS** A pointer to the created **BLK_DEV**, or NULL if parameters exceed physical device boundaries, if the physical device is not a block device, or if memory is insufficient for the structures.
- SEE ALSO** *scsiLib*

scsiBlkDevInit()

- NAME** *scsiBlkDevInit()* – initialize fields in a SCSI logical partition
- SYNOPSIS**
- ```
void scsiBlkDevInit
(
 SCSI_BLK_DEV * pScsiBlkDev, /* ptr to SCSI block dev. struct */
 int blksPerTrack, /* blocks per track */
 int nHeads /* number of heads */
)
```
- DESCRIPTION** This routine specifies the disk-geometry parameters required by certain file systems (for example, dosFs). It is called after a **SCSI\_BLK\_DEV** structure is created with *scsiBlkDevCreate()*, but before calling a file system initialization routine. It is generally required only for removable-media devices.

**RETURNS** N/A

**SEE ALSO** **scsiLib**

---

## ***scsiBlkDevShow()***

**NAME** *scsiBlkDevShow()* – show the BLK\_DEV structures on a specified physical device

**SYNOPSIS**

```
void scsiBlkDevShow
(
 SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device info */
)
```

**DESCRIPTION** This routine displays all of the BLK\_DEV structures created on a specified physical device. This routine is called by *scsiShow()* but may also be invoked directly, usually from the shell.

**RETURNS** N/A

**SEE ALSO** **scsiLib**, *scsiShow()*

---

## ***scsiBusReset()***

**NAME** *scsiBusReset()* – pulse the reset signal on the SCSI bus

**SYNOPSIS**

```
STATUS scsiBusReset
(
 SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */
)
```

**DESCRIPTION** This routine calls a controller-specific routine to reset a specified controller's SCSI bus. If no controller is specified (*pScsiCtrl* is 0), the value in the global variable **pSysScsiCtrl** is used.

**RETURNS** OK, or ERROR if there is no controller or controller-specific routine.

**SEE ALSO** **scsiLib**

---

## *scsiCacheSnoopDisable()*

|                    |                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>scsiCacheSnoopDisable()</i> – inform SCSI that hardware snooping of caches is disabled                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>void scsiCacheSnoopDisable (     SCSI_CTRL * pScsiCtrl /* pointer to a SCSI_CTRL structure */ )</pre>                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | This routine informs the SCSI library that hardware snooping is disabled and that <b>scsi2Lib</b> should execute any necessary cache coherency code. In order to make <b>scsi2Lib</b> aware that hardware snooping is disabled, this routine should be called after all SCSI-2 initializations, especially after <i>scsi2CtrlInit()</i> . |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>scsi2Lib</b>                                                                                                                                                                                                                                                                                                                           |

---

## *scsiCacheSnoopEnable()*

|                    |                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>scsiCacheSnoopEnable()</i> – inform SCSI that hardware snooping of caches is enabled                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>void scsiCacheSnoopEnable (     SCSI_CTRL * pScsiCtrl /* pointer to a SCSI_CTRL structure */ )</pre>                                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | This routine informs the SCSI library that hardware snooping is enabled and that <b>scsi2Lib</b> need not execute any cache coherency code. In order to make <b>scsi2Lib</b> aware that hardware snooping is enabled, this routine should be called after all SCSI-2 initializations, especially after <i>scsi2CtrlInit()</i> . |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <b>scsi2Lib</b>                                                                                                                                                                                                                                                                                                                 |

---

## *scsiCacheSynchronize()*

**NAME** *scsiCacheSynchronize()* – synchronize the caches for data coherency

**SYNOPSIS**

```
void scsiCacheSynchronize
(
 SCSI_THREAD * pThread, /* ptr to thread info */
 SCSI_CACHE_ACTION action /* cache action required */
)
```

**DESCRIPTION** This routine performs whatever cache action is necessary to ensure cache coherency with respect to the various buffers involved in a SCSI command.

The process is as follows:

1. The buffers for command, identification, and write data, which are simply written to SCSI, are flushed before the command.
2. The status buffer, which is written and then read, is cleared (flushed and invalidated) before the command.
3. The data buffer for a read command, which is only read, is cleared before the command.

The data buffer for a read command is cleared before the command rather than invalidated after it because it may share dirty cache lines with data outside the read buffer. DMA drivers for older versions of the SCSI library have flushed the first and last bytes of the data buffer before the command. However, this approach is not sufficient with the enhanced SCSI library because the amount of data transferred into the buffer may not fill it, which would cause dirty cache lines which contain correct data for the un-filled part of the buffer to be lost when the buffer is invalidated after the command.

To optimize the performance of the driver in supporting different caching policies, the routine uses the `CACHE_USER_FLUSH` macro when flushing the cache. In the absence of a `CACHE_USER_CLEAR` macro, the following steps are taken:

1. If there is a non-NULL flush routine in the `cacheUserFuncs` structure, the cache is cleared.
2. If there is a non-NULL invalidate routine, the cache is invalidated.
3. Otherwise nothing is done; the cache is assumed to be coherent without any software intervention.

Finally, since flushing (clearing) cache line entries for a large data buffer can be time-consuming, if the data buffer is larger than a preset (run-time configurable) size, the entire cache is flushed.

**RETURNS** N/A



SEE ALSO *scsi2Lib*

---

## *scsiErase()*

**NAME** *scsiErase()* – issue an ERASE command to a SCSI device

**SYNOPSIS**

```
STATUS scsiErase
(
 SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
 BOOL longErase /* TRUE for entire tape erase */
)
```

**DESCRIPTION** This routine issues an ERASE command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** *scsiSeqLib*

---

## *scsiFormatUnit()*

**NAME** *scsiFormatUnit()* – issue a FORMAT\_UNIT command to a SCSI device

**SYNOPSIS**

```
STATUS scsiFormatUnit
(
 SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
 BOOL cmpDefectList, /* whether defect list is complete */
 int defListFormat, /* defect list format */
 int vendorUnique, /* vendor unique byte */
 int interleave, /* interleave factor */
 char * buffer, /* ptr to input data buffer */
 int bufLength /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a FORMAT\_UNIT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** *scsiLib*

---

## ***scsiIdentMsgBuild()***

**NAME** *scsiIdentMsgBuild()* – build an identification message

**SYNOPSIS**

```
int scsiIdentMsgBuild
(
 UINT8 * msg,
 SCSI_PHYS_DEV * pScsiPhysDev,
 SCSI_TAG_TYPE tagType,
 UINT tagNumber
)
```

**DESCRIPTION** This routine builds an identification message in the caller's buffer, based on the specified physical device, tag type, and tag number.

If the target device does not support messages, there is no identification message to build.

Otherwise, the identification message consists of an IDENTIFY byte plus an optional QUEUE TAG message (two bytes), depending on the type of tag used.

**NOTE** This function is not intended for use by application programs.

**RETURNS** The length of the resulting identification message in bytes or -1 for ERROR.

**SEE ALSO** *scsi2Lib*

---

## ***scsiIdentMsgParse()***

**NAME** *scsiIdentMsgParse()* – parse an identification message

**SYNOPSIS**

```
SCSI_IDENT_STATUS scsiIdentMsgParse
(
 SCSI_CTRL * pScsiCtrl,
 UINT8 * msg,
 int msgLength,
 SCSI_PHYS_DEV * * ppScsiPhysDev,
 SCSI_TAG * pTagNum
)
```

**DESCRIPTION** This routine scans a (possibly incomplete) identification message, validating it in the process. If there is an IDENTIFY message, it identifies the corresponding physical device.

If the physical device is currently processing an untagged (ITL) nexus, identification is complete. Otherwise, the identification is complete only if there is a complete QUEUE TAG message.

If there is no physical device corresponding to the IDENTIFY message, or if the device is processing tagged (ITLQ) nexuses and the tag does not correspond to an active thread (it may have been aborted by a timeout, for example), then the identification sequence fails.

The caller's buffers for physical device and tag number (the results of the identification process) are always updated. This is required by the thread event handler (see *scsiMgrThreadEvent()*.)

- NOTE** This function is not intended for use by application programs.
- RETURNS** The identification status (incomplete, complete, or rejected).
- SEE ALSO** `scsi2Lib`

---

## ***scsiInquiry()***

- NAME** *scsiInquiry()* – issue an INQUIRY command to a SCSI device
- SYNOPSIS**
- ```
STATUS scsiInquiry
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    char *         buffer,        /* ptr to input data buffer */
    int            bufLength      /* length of buffer in bytes */
)
```
- DESCRIPTION** This routine issues an INQUIRY command to a specified SCSI device.
- RETURNS** OK, or ERROR if the command fails.
- SEE ALSO** `scsiLib`

scsiIoctl()

scsiIoctl()

NAME *scsiIoctl()* – perform a device-specific I/O control function

SYNOPSIS

```
STATUS scsiIoctl
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI block device info */
    int            function,      /* function code */
    int            arg            /* argument to pass called function */
)
```

DESCRIPTION This routine performs a specified **ioctl** function using a specified SCSI block device.

RETURNS The status of the request, or ERROR if the request is unsupported.

SEE ALSO *scsiLib*

scsiLoadUnit()

NAME *scsiLoadUnit()* – issue a LOAD/UNLOAD command to a SCSI device

SYNOPSIS

```
STATUS scsiLoadUnit
(
    SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI physical device */
    BOOL         load,          /* TRUE=load, FALSE=unload */
    BOOL         reten,        /* TRUE=retention and unload */
    BOOL         eot           /* TRUE=end of tape and unload */
)
```

DESCRIPTION This routine issues a LOAD/UNLOAD command to a specified SCSI device.

RETURNS OK, or ERROR if the command fails.

SEE ALSO *scsiSeqLib*

scsiMgrBusReset()

| | |
|--------------------|--|
| NAME | <i>scsiMgrBusReset()</i> – handle a controller-bus reset event |
| SYNOPSIS | <pre>void scsiMgrBusReset (SCSI_CTRL * pScsiCtrl /* SCSI ctrlr on which bus reset */)</pre> |
| DESCRIPTION | This routine resets in turn: each attached physical device, each target, and the controller-finite-state machine. In practice, this routine implements the SCSI hard reset option. |
| NOTE | This routine does not physically reset the SCSI bus; see <i>scsiBusReset()</i> . This routine should not be called by application programs. |
| RETURNS | N/A |
| SEE ALSO | <i>scsiMgrLib</i> |

scsiMgrCtrlEvent()

| | |
|--------------------|---|
| NAME | <i>scsiMgrCtrlEvent()</i> – send an event to the SCSI controller state machine |
| SYNOPSIS | <pre>void scsiMgrCtrlEvent (SCSI_CTRL * pScsiCtrl, SCSI_EVENT_TYPE eventType)</pre> |
| DESCRIPTION | This routine is called by the thread driver whenever selection, reselection, or disconnection occurs or when a thread is activated. It manages a simple finite-state machine for the SCSI controller. |
| NOTE | This function should not be called by application programs. |
| RETURNS | N/A |
| SEE ALSO | <i>scsiMgrLib</i> |

scsiMgrEventNotify()

- NAME** *scsiMgrEventNotify()* – notify the SCSI manager of a SCSI (controller) event
- SYNOPSIS**
- ```
STATUS scsiMgrEventNotify
(
 SCSI_CTRL * pScsiCtrl, /* pointer to SCSI controller structure */
 SCSI_EVENT * pEvent, /* pointer to the SCSI event */
 int eventSize /* size of the event information */
)
```
- DESCRIPTION** This routine posts an event message on the appropriate SCSI manager queue, then notifies the SCSI manager that there is a message to be accepted.
- NOTE** This routine should not be called by application programs.  
No access serialization is required, because event messages are only posted by the SCSI controller ISR. See the reference entry for *scsiBusResetNotify()*.
- RETURNS** OK, or ERROR if the SCSI manager's event queue is full.
- SEE ALSO** *scsiMgrLib*, *scsiBusResetNotify()*

---

## **scsiMgrShow()**

- NAME** *scsiMgrShow()* – show status information for the SCSI manager
- SYNOPSIS**
- ```
void scsiMgrShow
(
    SCSI_CTRL * pScsiCtrl, /* SCSI controller to use */
    BOOL      showPhysDevs, /* TRUE => show phys dev details */
    BOOL      showThreads,  /* TRUE => show thread details */
    BOOL      showFreeThreads /* TRUE => show free thread IDs */
)
```
- DESCRIPTION** This routine shows the current state of the SCSI manager for the specified controller, including the total number of threads created and the number of threads currently free.
Optionally, this routine also shows details for all created physical devices on this controller and all threads for which SCSI requests are outstanding. It also shows the IDs of all free threads.

- NOTE** The information displayed is volatile; this routine is best used when there is no activity on the SCSI bus. Threads allocated by a client but for which there are no outstanding SCSI requests are not shown.
- RETURNS** N/A
- SEE ALSO** *scsiMgrLib*

scsiMgrThreadEvent()

- NAME** *scsiMgrThreadEvent()* – send an event to the thread state machine
- SYNOPSIS**
- ```
void scsiMgrThreadEvent
(
 SCSI_THREAD * pThread,
 SCSI_THREAD_EVENT_TYPE eventType
)
```
- DESCRIPTION** This routine forwards an event to the thread’s physical device. If the event is completion or deferral, it frees up the tag which was allocated when the thread was activated and either completes or defers the thread.
- NOTE** This function should not be called by application programs.
- The thread passed into this function does not have to be an active client thread (it may be an identification thread).
- If the thread has no corresponding physical device, this routine does nothing. (This occasionally occurs if an unexpected disconnection or bus reset happens when an identification thread has not yet identified which physical device it corresponds to.
- RETURNS** N/A
- SEE ALSO** *scsiMgrLib*

***scsiModeSelect()***

---

***scsiModeSelect()***

**NAME** *scsiModeSelect()* – issue a MODE\_SELECT command to a SCSI device

**SYNOPSIS**

```
STATUS scsiModeSelect
(
 SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
 int pageFormat, /* value of the page format bit (0-1) */
 int saveParams, /* value of the save parameters bit (0-1) */
 char * buffer, /* ptr to output data buffer */
 int bufLength /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a MODE\_SELECT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** *scsiLib*

---

***scsiModeSense()***

**NAME** *scsiModeSense()* – issue a MODE\_SENSE command to a SCSI device

**SYNOPSIS**

```
STATUS scsiModeSense
(
 SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
 int pageControl, /* value of the page control field (0-3) */
 int pageCode, /* value of the page code field (0-0x3f) */
 char * buffer, /* ptr to input data buffer */
 int bufLength /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a MODE\_SENSE command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** *scsiLib*



---

## *scsiMsgInComplete()*

**NAME** *scsiMsgInComplete()* – handle a complete SCSI message received from the target

**SYNOPSIS**

```
STATUS scsiMsgInComplete
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 SCSI_THREAD * pThread /* ptr to thread info */
)
```

**DESCRIPTION** This routine parses the complete message and takes any necessary action, which may include setting up an outgoing message in reply. If the message is not understood, the routine rejects it and returns an ERROR status.

**NOTE** This function is intended for use only by SCSI controller drivers.

**RETURNS** OK, or ERROR if the message is not supported.

**SEE ALSO** *scsi2Lib*

---

## *scsiMsgOutComplete()*

**NAME** *scsiMsgOutComplete()* – perform post-processing after a SCSI message is sent

**SYNOPSIS**

```
STATUS scsiMsgOutComplete
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 SCSI_THREAD * pThread /* ptr to thread info */
)
```

**DESCRIPTION** This routine parses the complete message and takes any necessary action.

**NOTE** This function is intended for use only by SCSI controller drivers.

**RETURNS** OK, or ERROR if the message is not supported.

**SEE ALSO** *scsi2Lib*

***scsiMsgOutReject()***

---

***scsiMsgOutReject()***

**NAME** *scsiMsgOutReject()* – perform post-processing when an outgoing message is rejected

**SYNOPSIS**

```
void scsiMsgOutReject
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 SCSI_THREAD * pThread /* ptr to thread info */
)
```

**DESCRIPTIONNOTE** This function is intended for use only by SCSI controller drivers.

**RETURNS** OK, or ERROR if the message is not supported.

**SEE ALSO** *scsi2Lib*

---

***scsiPhysDevCreate()***

**NAME** *scsiPhysDevCreate()* – create a SCSI physical device structure

**SYNOPSIS**

```
SCSI_PHYS_DEV * scsiPhysDevCreate
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 int devBusId, /* device's SCSI bus ID */
 int devLUN, /* device's logical unit number */
 int reqSenseLength, /* length of REQUEST SENSE data dev returns */
 int devType, /* type of SCSI device */
 BOOL removable, /* whether medium is removable */
 int numBlocks, /* number of blocks on device */
 int blockSize /* size of a block in bytes */
)
```

**DESCRIPTION** This routine enables access to a SCSI device and must be the first routine invoked. It must be called once for each physical device on the SCSI bus.

If *reqSenseLength* is NULL (0), one or more **REQUEST\_SENSE** commands are issued to the device to determine the number of bytes of sense data it typically returns. Note that if the device returns variable amounts of sense data depending on its state, you must consult the device manual to determine the maximum amount of sense data that can be returned.

If *devType* is NONE (-1), an INQUIRY command is issued to determine the device type; as an added benefit, it acquires the device's make and model number. The *scsiShow()* routine displays this information. Common values of *devType* can be found in *scsiLib.h* or in the SCSI specification.

If *numBlocks* or *blockSize* are specified as NULL (0), a READ\_CAPACITY command is issued to determine those values. This occurs only for device types supporting READ\_CAPACITY.

**RETURNS** A pointer to the created SCSI\_PHYS\_DEV structure, or NULL if the routine is unable to create the physical-device structure.

**SEE ALSO** *scsiLib*

---

## *scsiPhysDevDelete()*

**NAME** *scsiPhysDevDelete()* – delete a SCSI physical-device structure

**SYNOPSIS**

```
STATUS scsiPhysDevDelete
(
 SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device info */
)
```

**DESCRIPTION** This routine deletes a specified SCSI physical-device structure.

**RETURNS** OK, or ERROR if *pScsiPhysDev* is NULL or SCSI\_BLK\_DEVS have been created on the device.

**SEE ALSO** *scsiLib*

---

## *scsiPhysDevIdGet()*

**NAME** *scsiPhysDevIdGet()* – return a pointer to a SCSI\_PHYS\_DEV structure

**SYNOPSIS**

```
SCSI_PHYS_DEV * scsiPhysDevIdGet
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 int devBusId, /* device's SCSI bus ID */
 int devLUN /* device's logical unit number */
)
```

***scsiPhysDevShow()***

|                    |                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine returns a pointer to the <code>SCSI_PHYS_DEV</code> structure of the SCSI physical device located at a specified bus ID ( <i>devBusId</i> ) and logical unit number ( <i>devLUN</i> ) and attached to a specified SCSI controller ( <i>pScsiCtrl</i> ). |
| <b>RETURNS</b>     | A pointer to the <code>SCSI_PHYS_DEV</code> structure, or NULL if the structure does not exist.                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <code>scsiLib</code>                                                                                                                                                                                                                                                 |

---

## ***scsiPhysDevShow()***

**NAME** *scsiPhysDevShow()* – show status information for a physical device

**SYNOPSIS**

```
void scsiPhysDevShow
(
 SCSI_PHYS_DEV * pScsiPhysDev, /* physical device to be displayed */
 BOOL showThreads, /* show IDs of associated threads */
 BOOL noHeader /* do not print title line */
)
```

**DESCRIPTION** This routine shows the state, the current nexus type, the current tag number, the number of tagged commands in progress, and the number of waiting and active threads for a SCSI physical device. Optionally, it shows the IDs of waiting and active threads, if any. This routine may be called at any time, but note that all of the information displayed is volatile.

**RETURNS** N/A

**SEE ALSO** `scsi2Lib`

---

## ***scsiRdSecs()***

**NAME** *scsiRdSecs()* – read sector(s) from a SCSI block device

**SYNOPSIS**

```
STATUS scsiRdSecs
(
 SCSI_BLK_DEV * pScsiBlkDev, /* ptr to SCSI block device info */
 int sector, /* sector number to be read */
 int numSecs, /* total sectors to be read */
 char * buffer /* ptr to input data buffer */
)
```

**DESCRIPTION** This routine reads the specified physical sector(s) from a specified physical device.

**RETURNS** OK, or ERROR if the sector(s) cannot be read.

**SEE ALSO** `scsiLib`

## *scsiRdTape()*

**NAME** `scsiRdTape()` – read bytes or blocks from a SCSI tape device

**SYNOPSIS**

```
int scsiRdTape
(
 SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device info */
 UINT count, /* total bytes or blocks to be read */
 char * buffer, /* ptr to input data buffer */
 BOOL fixedSize /* if variable size blocks */
)
```

**DESCRIPTION** This routine reads the specified number of bytes or blocks from a specified physical device. If the boolean *fixedSize* is true, then *numBytes* represents the number of blocks of size *blockSize*, defined in the `pScsiPhysDev` structure. If variable block sizes are used (*fixedSize* = FALSE), then *numBytes* represents the actual number of bytes to be read.

**RETURNS** Number of bytes or blocks actually read, 0 if EOF, or ERROR.

**SEE ALSO** `scsiSeqLib`

## *scsiReadCapacity()*

**NAME** `scsiReadCapacity()` – issue a READ\_CAPACITY command to a SCSI device

**SYNOPSIS**

```
STATUS scsiReadCapacity
(
 SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
 int * pLastLBA, /* where to return last logical block */
 int * pBlkLength /* where to return block length */
)
```

**DESCRIPTION** This routine issues a READ\_CAPACITY command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** **scsiLib**

---

## ***scsiRelease()***

**NAME** *scsiRelease()* – issue a RELEASE command to a SCSI device

**SYNOPSIS**

```
STATUS scsiRelease
(
 SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device */
)
```

**DESCRIPTION** This routine issues a RELEASE command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** **scsiDirectLib**

---

## ***scsiReleaseUnit()***

**NAME** *scsiReleaseUnit()* – issue a RELEASE UNIT command to a SCSI device

**SYNOPSIS**

```
STATUS scsiReleaseUnit
(
 SCSI_SEQ_DEV * pScsiSeqDev /* ptr to SCSI sequential device */
)
```

**DESCRIPTION** This routine issues a RELEASE UNIT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** **scsiSeqLib**

---

## *scsiReqSense()*

|                    |                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>scsiReqSense()</i> – issue a REQUEST_SENSE command to a SCSI device and read results                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiReqSense (     SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */     char *          buffer,        /* ptr to input data buffer */     int             bufLength      /* length of buffer in bytes */ )</pre> |
| <b>DESCRIPTION</b> | This routine issues a REQUEST_SENSE command to a specified SCSI device and reads the results.                                                                                                                                               |
| <b>RETURNS</b>     | OK, or ERROR if the command fails.                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <i>scsiLib</i>                                                                                                                                                                                                                              |

---

## *scsiReserve()*

|                    |                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>scsiReserve()</i> – issue a RESERVE command to a SCSI device                                      |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiReserve (     SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device */ )</pre> |
| <b>DESCRIPTION</b> | This routine issues a RESERVE command to a specified SCSI device.                                    |
| <b>RETURNS</b>     | OK, or ERROR if the command fails.                                                                   |
| <b>SEE ALSO</b>    | <i>scsiDirectLib</i>                                                                                 |

## ***scsiReserveUnit()***

**NAME** *scsiReserveUnit()* – issue a RESERVE UNIT command to a SCSI device

**SYNOPSIS**

```
STATUS scsiReserveUnit
(
 SCSI_SEQ_DEV * pScsiSeqDev /* ptr to SCSI sequential device */
)
```

**DESCRIPTION** This routine issues a RESERVE UNIT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** *scsiSeqLib*

---

## ***scsiRewind()***

**NAME** *scsiRewind()* – issue a REWIND command to a SCSI device

**SYNOPSIS**

```
STATUS scsiRewind
(
 SCSI_SEQ_DEV * pScsiSeqDev /* ptr to SCSI Sequential device */
)
```

**DESCRIPTION** This routine issues a REWIND command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** *scsiSeqLib*



---

## *scsiSeqDevCreate()*

**NAME** *scsiSeqDevCreate()* – create a SCSI sequential device

**SYNOPSIS**

```
SEQ_DEV *scsiSeqDevCreate
(
 SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device info */
)
```

**DESCRIPTION** This routine creates a SCSI sequential device and saves a pointer to this SEQ\_DEV in the SCSI physical device. The following functions are initialized in this structure:

|                                 |                                           |
|---------------------------------|-------------------------------------------|
| <code>sd_seqRd</code>           | – <i>scsiRdTape()</i>                     |
| <code>sd_seqWrt</code>          | – <i>scsiWrtTape()</i>                    |
| <code>sd_ioctl</code>           | – <i>scsiIoctl()</i> (in <i>scsiLib</i> ) |
| <code>sd_seqWrtFileMarks</code> | – <i>scsiWrtFileMarks()</i>               |
| <code>sd_statusChk</code>       | – <i>scsiSeqStatusCheck()</i>             |
| <code>sd_reset</code>           | – (not used)                              |
| <code>sd_rewind</code>          | – <i>scsiRewind()</i>                     |
| <code>sd_reserve</code>         | – <i>scsiReserve()</i>                    |
| <code>sd_release</code>         | – <i>scsiRelease()</i>                    |
| <code>sd_readBlkLim</code>      | – <i>scsiSeqReadBlockLimits()</i>         |
| <code>sd_load</code>            | – <i>scsiLoadUnit()</i>                   |
| <code>sd_space</code>           | – <i>scsiSpace()</i>                      |
| <code>sd_erase</code>           | – <i>scsiErase()</i>                      |

Only one SEQ\_DEV per SCSI\_PHYS\_DEV is allowed, unlike BLK\_DEVS where an entire list is maintained. Therefore, this routine can be called only once per creation of a sequential device.

**RETURNS** A pointer to the SEQ\_DEV structure, or NULL if the command fails.

**SEE ALSO** *scsiSeqLib*

***scsiSeqIoctl()***

---

***scsiSeqIoctl()***

|                    |                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>scsiSeqIoctl()</i> – perform an I/O control function for sequential access devices                                                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>int scsiSeqIoctl (     SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device */     int          function,      /* ioctl function code */     int          arg            /* argument to pass to called function */ )</pre> |
| <b>DESCRIPTION</b> | This routine issues <b>scsiSeqLib</b> commands to perform sequential device-specific I/O control operations.                                                                                                                            |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                            |
| <b>ERRNO</b>       | S_scsiLib_INVALID_BLOCK_SIZE                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <b>scsiSeqLib</b>                                                                                                                                                                                                                       |

---

***scsiSeqReadBlockLimits()***

|                    |                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>scsiSeqReadBlockLimits()</i> – issue a READ_BLOCK_LIMITS command to a SCSI device                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiSeqReadBlockLimits (     SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device */     int *         pMaxBlockLength, /* where to return max block length */     UINT16 *     pMinBlockLength /* where to return min block length */ )</pre> |
| <b>DESCRIPTION</b> | This routine issues a <b>READ_BLOCK_LIMITS</b> command to a specified SCSI device.                                                                                                                                                                                 |
| <b>RETURNS</b>     | OK, or ERROR if the command fails.                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>scsiSeqLib</b>                                                                                                                                                                                                                                                  |

---

## *scsiSeqStatusCheck()*

|                    |                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>scsiSeqStatusCheck()</i> – detect a change in media                                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiSeqStatusCheck (     SCSI_SEQ_DEV * pScsiSeqDev /* ptr to a sequential dev */ )</pre>                                                                                    |
| <b>DESCRIPTION</b> | This routine issues a <code>TEST_UNIT_READY</code> command to a SCSI device to detect a change in media. It is called by file systems before executing <i>open()</i> or <i>creat()</i> . |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <i>scsiSeqLib</i>                                                                                                                                                                        |

---

## *scsiShow()*

|                    |                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>scsiShow()</i> – list the physical devices attached to a SCSI controller                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiShow (     SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */ )</pre>                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | This routine displays the SCSI bus ID, logical unit number (LUN), vendor ID, product ID, firmware revision (rev.), device type, number of blocks, block size in bytes, and a pointer to the associated <code>SCSI_PHYS_DEV</code> structure for each physical SCSI device known to be attached to a specified SCSI controller. |
| <b>NOTE</b>        | If <i>pScsiCtrl</i> is NULL, the value of the global variable <code>pSysScsiCtrl</code> is used, unless it is also NULL.                                                                                                                                                                                                       |
| <b>RETURNS</b>     | OK, or ERROR if both <i>pScsiCtrl</i> and <code>pSysScsiCtrl</code> are NULL.                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <i>scsiLib</i>                                                                                                                                                                                                                                                                                                                 |

***scsiSpace()***

---

***scsiSpace()*****NAME** *scsiSpace()* – move the tape on a specified physical SCSI device**SYNOPSIS**  

```
STATUS scsiSpace
(
 SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device info */
 int count, /* count for space command */
 int spaceCode /* code for the type of space command */
)
```

**DESCRIPTION** This routine moves the tape on a specified SCSI physical device. There are two types of space code that are mandatory in SCSI; currently these are the only two supported:

| Code | Description           | Support |
|------|-----------------------|---------|
| 000  | Blocks                | Yes     |
| 001  | File marks            | Yes     |
| 010  | Sequential file marks | No      |
| 011  | End-of-data           | No      |
| 100  | Set marks             | No      |
| 101  | Sequential set marks  | No      |

**RETURNS** OK, or ERROR if an error is returned by the device.**ERRNO** S\_scsiLib\_ILLEGAL\_REQUEST**SEE ALSO** *scsiSeqLib*

---

***scsiStartStopUnit()*****NAME** *scsiStartStopUnit()* – issue a START\_STOP\_UNIT command to a SCSI device**SYNOPSIS**  

```
STATUS scsiStartStopUnit
(
 SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
 BOOL start /* TRUE == start, FALSE == stop */
)
```

**DESCRIPTION** This routine issues a START\_STOP\_UNIT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** *scsiDirectLib*

2

---

## *scsiSyncXferNegotiate()*

**NAME** *scsiSyncXferNegotiate()* – initiate or continue negotiating transfer parameters

**SYNOPSIS**

```
void scsiSyncXferNegotiate
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 SCSI_TARGET * pScsiTarget, /* ptr to SCSI target info */
 SCSI_SYNC_XFER_EVENT eventType /* tells what has just happened */
)
```

**DESCRIPTION** This routine manages negotiation by means of a finite-state machine which is driven by "significant events" such as incoming and outgoing messages. Each SCSI target has its own independent state machine.

**NOTE** If the controller does not support synchronous transfer or if the target's maximum REQ/ACK offset is zero, attempts to initiate a round of negotiation are ignored. This function is intended for use only by SCSI controller drivers.

**RETURNS** N/A

**SEE ALSO** *scsi2Lib*

---

## *scsiTapeModeSelect()*

**NAME** *scsiTapeModeSelect()* – issue a MODE\_SELECT command to a SCSI tape device

**SYNOPSIS**

```
STATUS scsiTapeModeSelect
(
 SCSI_PHYS_DEV *pScsiPhysDev, /* ptr to SCSI physical device */
 int pageFormat, /* value of the page format bit (0-1) */
 int saveParams, /* value of the save parameters bit (0-1) */
 char *buffer, /* ptr to output data buffer */
 int bufLength /* length of buffer in bytes */
)
```

***scsiTapeModeSense()***

**DESCRIPTION** This routine issues a **MODE\_SELECT** command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiSeqLib`

---

***scsiTapeModeSense()***

**NAME** `scsiTapeModeSense()` – issue a **MODE\_SENSE** command to a SCSI tape device

**SYNOPSIS**

```
STATUS scsiTapeModeSense
(
 SCSI_PHYS_DEV *pScsiPhysDev, /* ptr to SCSI physical device */
 int pageControl, /* value of the page control field (0-3) */
 int pageCode, /* value of the page code field (0-0x3f) */
 char *buffer, /* ptr to input data buffer */
 int bufLength /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a **MODE\_SENSE** command to a specified SCSI tape device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiSeqLib`

---

***scsiTargetOptionsGet()***

**NAME** `scsiTargetOptionsGet()` – get options for one or all SCSI targets

**SYNOPSIS**

```
STATUS scsiTargetOptionsGet
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 int devBusId, /* target to interrogate */
 SCSI_OPTIONS * pOptions /* buffer to return options */
)
```

**DESCRIPTION** This routine copies the current options for the specified target into the caller's buffer.

**RETURNS** OK, or ERROR if the bus ID is invalid.

SEE ALSO *scsi2Lib*

---

## *scsiTargetOptionsSet()*

**NAME** *scsiTargetOptionsSet()* – set options for one or all SCSI targets

**SYNOPSIS**

```
STATUS scsiTargetOptionsSet
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 int devBusId, /* target to affect, or all */
 SCSI_OPTIONS * pOptions, /* buffer containing new options */
 UINT which /* which options to change */
)
```

**DESCRIPTION** This routine sets the options defined by the bitmask **which** for the specified target (or all targets if **devBusId** is `SCSI_SET_OPT_ALL_TARGETS`).

The bitmask **which** can be any combination of the following, bitwise OR'd together (corresponding fields in the `SCSI_OPTIONS` structure are shown in parentheses):

|                                       |                                       |                                                                        |
|---------------------------------------|---------------------------------------|------------------------------------------------------------------------|
| <code>SCSI_SET_OPT_TIMEOUT</code>     | <b>selTimeOut</b>                     | select timeout period, microseconds                                    |
| <code>SCSI_SET_OPT_MESSAGES</code>    | <b>messages</b>                       | FALSE to disable SCSI messages                                         |
| <code>SCSI_SET_OPT_DISCONNECT</code>  | <b>disconnect</b>                     | FALSE to disable discon/recon                                          |
| <code>SCSI_SET_OPT_XFER_PARAMS</code> | <b>maxOffset,</b><br><b>minPeriod</b> | max sync xfer offset, 0>async<br>min sync xfer period, x 4 nsec.       |
| <code>SCSI_SET_OPT_TAG_PARAMS</code>  | <b>tagType,</b><br><b>maxTags</b>     | default tag type ( <code>SCSI_TAG_*</code> )<br>max cmd tags available |
| <code>SCSI_SET_OPT_WIDE_PARAMS</code> | <b>xferWidth</b>                      | data transfer width in bits                                            |

**NOTE** This routine can be used after the target device has already been used; in this case, however, it is not possible to change the tag parameters. This routine must not be used while there is any SCSI activity on the specified target(s).

**RETURNS** OK, or ERROR if the bus ID or options are invalid.

SEE ALSO *scsi2Lib*

***scsiTestUnitRdy()***

---

***scsiTestUnitRdy()***

**NAME** *scsiTestUnitRdy()* – issue a TEST\_UNIT\_READY command to a SCSI device

**SYNOPSIS** `STATUS scsiTestUnitRdy`  
`(`  
`SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device */`  
`)`

**DESCRIPTION** This routine issues a TEST\_UNIT\_READY command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiLib`

---

***scsiThreadInit()***

**NAME** *scsiThreadInit()* – perform generic SCSI thread initialization

**SYNOPSIS** `STATUS scsiThreadInit`  
`(`  
`SCSI_THREAD * pThread`  
`)`

**DESCRIPTION** This routine initializes the controller-independent parts of a thread structure, which are specific to the SCSI manager.

**NOTE** This function should not be called by application programs. It is intended to be used by SCSI controller drivers.

**RETURNS** OK, or ERROR if the thread cannot be initialized.

**SEE ALSO** `scsi2Lib`



---

## *scsiWideXferNegotiate()*

**NAME** *scsiWideXferNegotiate()* – initiate or continue negotiating wide parameters

**SYNOPSIS**

```
void scsiWideXferNegotiate
(
 SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
 SCSI_TARGET * pScsiTarget, /* ptr to SCSI target info */
 SCSI_WIDE_XFER_EVENT eventType /* tells what has just happened */
)
```

**DESCRIPTION** This routine manages negotiation means of a finite-state machine which is driven by "significant events" such as incoming and outgoing messages. Each SCSI target has its own independent state machine.

**NOTE** If the controller does not support wide transfers or the target's transfer width is zero, attempts to initiate a round of negotiation are ignored; this is because zero is the default narrow transfer.

This function is intended for use only by SCSI controller drivers.

**RETURNS** N/A

**SEE ALSO** *scsi2Lib*

---

## *scsiWrtFileMarks()*

**NAME** *scsiWrtFileMarks()* – write file marks to a SCSI sequential device

**SYNOPSIS**

```
STATUS scsiWrtFileMarks
(
 SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device info */
 int numMarks, /* number of file marks to write */
 BOOL shortMark /* TRUE to write short file mark */
)
```

**DESCRIPTION** This routine writes file marks to a specified physical device.

**RETURNS** OK, or ERROR if the file mark cannot be written.

**SEE ALSO** *scsiSeqLib*

**scsiWrtSecs()**

---

**scsiWrtSecs()**

**NAME** `scsiWrtSecs()` – write sector(s) to a SCSI block device

**SYNOPSIS**

```
STATUS scsiWrtSecs
(
 SCSI_BLK_DEV * pScsiBlkDev, /* ptr to SCSI block device info */
 int sector, /* sector number to be written */
 int numSecs, /* total sectors to be written */
 char * buffer /* ptr to input data buffer */
)
```

**DESCRIPTION** This routine writes the specified physical sector(s) to a specified physical device.

**RETURNS** OK, or ERROR if the sector(s) cannot be written.

**SEE ALSO** `scsiLib`

---

**scsiWrtTape()**

**NAME** `scsiWrtTape()` – write data to a SCSI tape device

**SYNOPSIS**

```
STATUS scsiWrtTape
(
 SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device info */
 int numBytes, /* total bytes or blocks to be written */
 char * buffer, /* ptr to input data buffer */
 BOOL fixedSize /* if variable size blocks */
)
```

**DESCRIPTION** This routine writes data to the current block on a specified physical device. If the boolean *fixedSize* is true, then *numBytes* represents the number of blocks of size *blockSize*, defined in the `pScsiPhysDev` structure. If variable block sizes are used (*fixedSize* = FALSE), then *numBytes* represents the actual number of bytes to be written. If *numBytes* is greater than the `maxBytesLimit` field defined in the `pScsiPhysDev` structure, then more than one SCSI transaction is used to transfer the data.

**RETURNS** OK, or ERROR if the data cannot be written or zero bytes are written.

**SEE ALSO** `scsiSeqLib`

---

## *select()*

**NAME** *select()* – pend on a set of file descriptors

**SYNOPSIS**

```
int select
(
 int width, /* number of bits to examine from 0 */
 fd_set * pReadFds, /* read fds */
 fd_set * pWriteFds, /* write fds */
 fd_set * pExceptFds, /* exception fds (unsupported) */
 struct timeval * pTimeout /* max time to wait, NULL = forever */
)
```

**DESCRIPTION** This routine permits a task to pend until one of a set of file descriptors becomes ready. Three parameters -- *pReadFds*, *pWriteFds*, and *pExceptFds* -- point to file descriptor sets in which each bit corresponds to a particular file descriptor. Bits set in the read file descriptor set (*pReadFds*) will cause *select()* to pend until data is available on any of the corresponding file descriptors, while bits set in the write file descriptor set (*pWriteFds*) will cause *select()* to pend until any of the corresponding file descriptors become writable. (The *pExceptFds* parameter is currently unused, but is provided for UNIX call compatibility.)

The following macros are available for setting the appropriate bits in the file descriptor set structure:

```
FD_SET(fd, &fdset)
FD_CLR(fd, &fdset)
FD_ZERO(&fdset)
```

If either *pReadFds* or *pWriteFds* is NULL, they are ignored. The *width* parameter defines how many bits will be examined in the file descriptor sets, and should be set to either the maximum file descriptor value in use plus one, or simply to `FD_SETSIZE`. When *select()* returns, it zeros out the file descriptor sets, and sets only the bits that correspond to file descriptors that are ready. The `FD_ISSET` macro may be used to determine which bits are set.

If *pTimeout* is NULL, *select()* will block indefinitely. If *pTimeout* is not NULL, but points to a `timeval` structure with an effective time of zero, the file descriptors in the file descriptor sets will be polled and the results returned immediately. If the effective time value is greater than zero, *select()* will return after the specified time has elapsed, even if none of the file descriptors are ready.

Applications can use *select()* with pipes and serial devices, in addition to sockets. Also, *select()* now examines write file descriptors in addition to read file descriptors; however, exception file descriptors remain unsupported.

## ***selectInit()***

Driver developers should consult the *VxWorks Programmer's Guide: I/O System* for details on writing drivers that will use *select()*.

**RETURNS** The number of file descriptors with activity, 0 if timed out, or ERROR if an error occurred when the driver's *select()* routine was invoked via *ioctl()*.

**SEE ALSO** *selectLib*, *VxWorks Programmer's Guide: I/O System*

---

## ***selectInit()***

**NAME** *selectInit()* – initialize the select facility

**SYNOPSIS** `void selectInit (void)`

**DESCRIPTION** This routine initializes the UNIX BSD 4.3 select facility. It should be called only once, and typically is called from the root task, *usrRoot()*, in *usrConfig.c*. It installs a task delete hook that cleans up after a task if the task is deleted while pending in *select()*.

**RETURNS** N/A

**SEE ALSO** *selectLib*

---

## ***selNodeAdd()***

**NAME** *selNodeAdd()* – add a wake-up node to a *select()* wake-up list

**SYNOPSIS** `STATUS selNodeAdd  
(  
 SEL_WAKEUP_LIST * pWakeupList, /* list of tasks to wake up */  
 SEL_WAKEUP_NODE * pWakeupNode /* node to add to list */  
)`

**DESCRIPTION** This routine adds a wake-up node to a device's wake-up list. It is typically called from a driver's FIOSELECT function.

**RETURNS** OK, or ERROR if memory is insufficient.

**SEE ALSO** *selectLib*

---

## *selNodeDelete()*

|                    |                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>selNodeDelete()</i> – find and delete a node from a <i>select()</i> wake-up list                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS selNodeDelete (     SEL_WAKEUP_LIST * pWakeupList, /* list of tasks to wake up */     SEL_WAKEUP_NODE * pWakeupNode /* node to delete from list */ )</pre> |
| <b>DESCRIPTION</b> | This routine deletes a specified wake-up node from a specified wake-up list. Typically, it is called by a driver's FIOUNSELECT function.                               |
| <b>RETURNS</b>     | OK, or ERROR if the node is not found in the wake-up list.                                                                                                             |
| <b>SEE ALSO</b>    | <i>selectLib</i>                                                                                                                                                       |

---

## *selWakeup()*

|                    |                                                                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>selWakeup()</i> – wake up a task pending in <i>select()</i>                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>void selWakeup (     SEL_WAKEUP_NODE * pWakeupNode /* node to wake up */ )</pre>                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | This routine wakes up a task pending in <i>select()</i> . Once a driver's FIOSELECT function installs a wake-up node in a device's wake-up list (using <i>selNodeAdd()</i> ) and checks to make sure the device is ready, this routine ensures that the <i>select()</i> call does not pend. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <i>selectLib</i>                                                                                                                                                                                                                                                                            |

---

## ***selWakeupAll()***

|                    |                                                                                                                                                                                                                                                                         |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>selWakeupAll()</i> – wake up all tasks in a <i>select()</i> wake-up list                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>void selWakeupAll (     SEL_WAKEUP_LIST * pWakeupList, /* list of tasks to wake up */     SELECT_TYPE      type          /* readers (SELREAD) or writers (SELWRITE) */ )</pre>                                                                                     |
| <b>DESCRIPTION</b> | This routine wakes up all tasks pended in <i>select()</i> that are waiting for a device; it is called by a driver when the device becomes ready. The <i>type</i> parameter specifies the task to be awakened, either reader tasks (SELREAD) or writer tasks (SELWRITE). |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <i>selectLib</i>                                                                                                                                                                                                                                                        |

---

## ***selWakeupListInit()***

|                    |                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>selWakeupListInit()</i> – initialize a <i>select()</i> wake-up list                                   |
| <b>SYNOPSIS</b>    | <pre>void selWakeupListInit (     SEL_WAKEUP_LIST * pWakeupList /* wake-up list to initialize */ )</pre> |
| <b>DESCRIPTION</b> | This routine should be called in a device's create routine to initialize the SEL_WAKEUP_LIST structure.  |
| <b>RETURNS</b>     | N/A                                                                                                      |
| <b>SEE ALSO</b>    | <i>selectLib</i>                                                                                         |

---

## *selWakeupListLen()*

|                    |                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>selWakeupListLen()</i> – get the number of nodes in a <i>select()</i> wake-up list                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>int selWakeupListLen (     SEL_WAKEUP_LIST * pWakeupList /* list of tasks to wake up */ )</pre>                                                                                                                                                             |
| <b>DESCRIPTION</b> | This routine returns the number of nodes in a specified SEL_WAKEUP_LIST. It can be used by a driver to determine if any tasks are currently pending in <i>select()</i> on this device, and whether these tasks need to be activated with <i>selWakeupAll()</i> . |
| <b>RETURNS</b>     | The number of nodes currently in a <i>select()</i> wake-up list, or ERROR.                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <i>selectLib</i>                                                                                                                                                                                                                                                 |

---

## *selWakeupType()*

|                    |                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>selWakeupType()</i> – get the type of a <i>select()</i> wake-up node                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>SELECT_TYPE selWakeupType (     SEL_WAKEUP_NODE * pWakeupNode /* node to get type of */ )</pre>                                                                                           |
| <b>DESCRIPTION</b> | This routine returns the type of a specified SEL_WAKEUP_NODE. It is typically used in a device's FIOSELECT function to determine if the device is being selected for read or write operations. |
| <b>RETURNS</b>     | SELREAD (read operation) or SELWRITE (write operation).                                                                                                                                        |
| <b>SEE ALSO</b>    | <i>selectLib</i>                                                                                                                                                                               |

---

## ***semBCreate()***

- NAME** *semBCreate()* – create and initialize a binary semaphore
- SYNOPSIS**
- ```
SEM_ID semBCreate
(
    int          options,      /* semaphore options */
    SEM_B_STATE initialState /* initial semaphore state */
)
```
- DESCRIPTION**
- This routine allocates and initializes a binary semaphore. The semaphore is initialized to the *initialState* of either **SEM_FULL** (1) or **SEM_EMPTY** (0).
- The *options* parameter specifies the queuing style for blocked tasks. Tasks can be queued on a priority basis or a first-in-first-out basis. These options are **SEM_Q_PRIORITY** (0x1) and **SEM_Q_FIFO** (0x0), respectively.
- RETURNS**
- The semaphore ID, or NULL if memory cannot be allocated.
- SEE ALSO**
- semBLib**

semBSmCreate()

- NAME** *semBSmCreate()* – create and initialize a shared memory binary semaphore (VxMP Opt.)
- SYNOPSIS**
- ```
SEM_ID semBSmCreate
(
 int options, /* semaphore options */
 SEM_B_STATE initialState /* initial semaphore state */
)
```
- DESCRIPTION**
- This routine allocates and initializes a shared memory binary semaphore. The semaphore is initialized to an *initialState* of either **SEM\_FULL** (available) or **SEM\_EMPTY** (not available). The shared semaphore structure is allocated from the shared semaphore dedicated memory partition.
- The semaphore ID returned by this routine can be used directly by the generic semaphore-handling routines in **semLib** -- *semGive()*, *semTake()*, and *semFlush()* -- and the show routines, such as *show()* and *semShow()*.
- The queuing style for blocked tasks is set by *options*; the only supported queuing style for shared memory semaphores is first-in-first-out, selected by **SEM\_Q\_FIFO**.



Before this routine can be called, the shared memory objects facility must be initialized (see **semSmLib**).

The maximum number of shared memory semaphores (binary plus counting) that can be created is **SM\_OBJ\_MAX\_SEM**.

|                     |                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory support option, VxMP.                                         |
| <b>RETURNS</b>      | The semaphore ID, or NULL if memory cannot be allocated from the shared semaphore dedicated memory partition.                           |
| <b>ERRNO</b>        | <b>S_memLib_NOT_ENOUGH_MEMORY</b> , <b>S_semLib_INVALID_QUEUE_TYPE</b> , <b>S_semLib_INVALID_STATE</b> , <b>S_smObjLib_LOCK_TIMEOUT</b> |
| <b>SEE ALSO</b>     | <b>semSmLib</b> , <b>semLib</b> , <b>semBLib</b> , <b>smObjLib</b> , <b>semShow</b>                                                     |

---

## ***semCCreate()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b><i>semCCreate()</i></b> – create and initialize a counting semaphore                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>SEM_ID semCCreate (     int options,      /* semaphore option modes */     int initialCount /* initial count */ )</pre>                                                                                                                                                                                                                                                        |
| <b>DESCRIPTION</b> | <p>This routine allocates and initializes a counting semaphore. The semaphore is initialized to the specified initial count.</p> <p>The <i>options</i> parameter specifies the queuing style for blocked tasks. Tasks may be queued on a priority basis or a first-in-first-out basis. These options are <b>SEM_Q_PRIORITY</b> (0x1) and <b>SEM_Q_FIFO</b> (0x0), respectively.</p> |
| <b>RETURNS</b>     | The semaphore ID, or NULL if memory cannot be allocated.                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <b>semCLib</b>                                                                                                                                                                                                                                                                                                                                                                      |

## ***semClear()***

- NAME** *semClear()* – take a release 4.x semaphore, if the semaphore is available
- SYNOPSIS**
- ```
STATUS semClear
(
    SEM_ID semId /* semaphore ID to empty */
)
```
- DESCRIPTION** This routine takes a VxWorks 4.x semaphore if it is available (full), otherwise no action is taken except to return ERROR. This routine never preempts the caller.
- RETURNS** OK, or ERROR if the semaphore is unavailable.
- SEE ALSO** semOLib
-

semCreate()

- NAME** *semCreate()* – create and initialize a release 4.x binary semaphore
- SYNOPSIS**
- ```
SEM_ID semCreate (void)
```
- DESCRIPTION** This routine allocates a VxWorks 4.x binary semaphore. The semaphore is initialized to empty. After initialization, it must be given before it can be taken.
- RETURNS** The semaphore ID, or NULL if memory cannot be allocated.
- SEE ALSO** semOLib, *semInit()*
- 

## ***semCSmCreate()***

- NAME** *semCSmCreate()* – create and initialize a shared memory counting semaphore (VxMP Opt.)
- SYNOPSIS**
- ```
SEM_ID semCSmCreate
(
    int options,      /* semaphore options */
    int initialCount /* initial semaphore count */
)
```

| | |
|---------------------|--|
| DESCRIPTION | <p>This routine allocates and initializes a shared memory counting semaphore. The initial count value of the semaphore (the number of times the semaphore should be taken before it can be given) is specified by <i>initialCount</i>.</p> <p>The semaphore ID returned by this routine can be used directly by the generic semaphore-handling routines in semLib -- <i>semGive()</i>, <i>semTake()</i> and <i>semFlush()</i> -- and the show routines, such as <i>show()</i> and <i>semShow()</i>.</p> <p>The queuing style for blocked tasks is set by <i>options</i>; the only supported queuing style for shared memory semaphores is first-in-first-out, selected by SEM_Q_FIFO.</p> <p>Before this routine can be called, the shared memory objects facility must be initialized (see semSmLib).</p> <p>The maximum number of shared memory semaphores (binary plus counting) that can be created is SM_OBJ_MAX_SEM.</p> |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory support option, VxMP. |
| RETURNS | The semaphore ID, or NULL if memory cannot be allocated from the shared semaphore dedicated memory partition. |
| ERRNO | S_memLib_NOT_ENOUGH_MEMORY , S_semLib_INVALID_QUEUE_TYPE , S_smObjLib_LOCK_TIMEOUT |
| SEE ALSO | semSmLib , semLib , semCLib , smObjLib , semShow |

semDelete()

| | |
|--------------------|---|
| NAME | <i>semDelete()</i> – delete a semaphore |
| SYNOPSIS | <pre>STATUS semDelete (SEM_ID semId /* semaphore ID to delete */)</pre> |
| DESCRIPTION | This routine terminates and deallocates any memory associated with a specified semaphore. Any pending tasks will unblock and return ERROR. |
| WARNING | Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that already has taken (owns) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully taken. |

semFlush()

- RETURNS** OK, or ERROR if the semaphore ID is invalid.
- ERRNO** S_intLib_NOT_ISR_CALLABLE, S_objLib_OBJ_ID_ERROR, S_smObjLib_NO_OBJECT_DESTROY
- SEE ALSO** semLib, semBLib, semCLib, semMLib, semSmLib

semFlush()

- NAME** *semFlush()* – unblock every task pended on a semaphore
- SYNOPSIS**
- ```
STATUS semFlush
(
 SEM_ID semId /* semaphore ID to unblock everyone for */
)
```
- DESCRIPTION**
- This routine atomically unblocks all tasks pended on a specified semaphore, i.e., all tasks will be unblocked before any is allowed to run. The state of the underlying semaphore is unchanged. All pended tasks will enter the ready queue before having a chance to execute.
- The flush operation is useful as a means of broadcast in synchronization applications. Its use is illegal for mutual-exclusion semaphores created with *semMCreate()*.
- RETURNS** OK, or ERROR if the semaphore ID is invalid or the operation is not supported.
- ERRNO** S\_objLib\_OBJ\_ID\_ERROR
- SEE ALSO** semLib, semBLib, semCLib, semMLib, semSmLib

---

## ***semGive()***

- NAME** *semGive()* – give a semaphore
- SYNOPSIS**
- ```
STATUS semGive
(
    SEM_ID semId /* semaphore ID to give */
)
```

| | |
|--------------------|---|
| DESCRIPTION | This routine performs the give operation on a specified semaphore. Depending on the type of semaphore, the state of the semaphore and of the pending tasks may be affected. The behavior of <i>semGive()</i> is discussed fully in the library description of the specific semaphore type being used. |
| RETURNS | OK, or ERROR if the semaphore ID is invalid. |
| ERRNO | S_intLib_NOT_ISR_CALLABLE, S_objLib_OBJ_ID_ERROR, S_semLib_INVALID_OPERATION |
| SEE ALSO | semLib, semBLib, semCLib, semMLib, semSmLib |

semInfo()

| | |
|--------------------|--|
| NAME | <i>semInfo()</i> – get a list of task IDs that are blocked on a semaphore |
| SYNOPSIS | <pre>int semInfo (SEM_ID semId, /* semaphore ID to summarize */ int idList[], /* array of task IDs to be filled in */ int maxTasks /* max tasks idList can accommodate */)</pre> |
| DESCRIPTION | This routine reports the tasks blocked on a specified semaphore. Up to <i>maxTasks</i> task IDs are copied to the array specified by <i>idList</i> . The array is unordered. |
| WARNING | There is no guarantee that all listed tasks are still valid or that new tasks have not been blocked by the time <i>semInfo()</i> returns. |
| RETURNS | The number of blocked tasks placed in <i>idList</i> . |
| SEE ALSO | semShow |

semInit()

semInit()

| | |
|--------------------|--|
| NAME | <i>semInit()</i> – initialize a static binary semaphore |
| SYNOPSIS | <pre>STATUS semInit (SEMAPHORE * pSemaphore /* 4.x semaphore to initialize */)</pre> |
| DESCRIPTION | This routine initializes static VxWorks 4.x semaphores. In some instances, a semaphore cannot be created with <i>semCreate()</i> but is a static object. |
| RETURNS | OK, or ERROR if the semaphore cannot be initialized. |
| SEE ALSO | <i>semOLib</i> , <i>semCreate()</i> |

semMCreate()

| | |
|--------------------|--|
| NAME | <i>semMCreate()</i> – create and initialize a mutual-exclusion semaphore |
| SYNOPSIS | <pre>SEM_ID semMCreate (int options /* mutex semaphore options */)</pre> |
| DESCRIPTION | <p>This routine allocates and initializes a mutual-exclusion semaphore. The semaphore state is initialized to full.</p> <p>Semaphore options include the following:</p> <p>SEM_Q_PRIORITY (0x1) Queue pending tasks on the basis of their priority.</p> <p>SEM_Q_FIFO (0x0) Queue pending tasks on a first-in-first-out basis.</p> <p>SEM_DELETE_SAFE (0x4) Protect a task that owns the semaphore from unexpected deletion. This option enables an implicit <i>taskSafe()</i> for each <i>semTake()</i>, and an implicit <i>taskUnsafe()</i> for each <i>semGive()</i>.</p> <p>SEM_INVERSION_SAFE (0x8) Protect the system from priority inversion. With this option, the task owning the</p> |

semaphore will execute at the highest priority of the tasks pended on the semaphore, if it is higher than its current priority. This option must be accompanied by the `SEM_Q_PRIORITY` queuing mode.

RETURNS The semaphore ID, or NULL if memory cannot be allocated.

SEE ALSO `semMLib`, `semLib`, `semBLib`, `taskSafe()`, `taskUnsafe()`

semMGiveForce()

NAME `semMGiveForce()` – give a mutual-exclusion semaphore without restrictions

SYNOPSIS

```
STATUS semMGiveForce
(
    SEM_ID semId /* semaphore ID to give */
)
```

DESCRIPTION This routine gives a mutual-exclusion semaphore, regardless of semaphore ownership. It is intended as a debugging aid only.

The routine is particularly useful when a task dies while holding some mutual-exclusion semaphore, because the semaphore can be resurrected. The routine will give the semaphore to the next task in the pend queue or make the semaphore full if no tasks are pending. In effect, execution will continue as if the task owning the semaphore had actually given the semaphore.

CAVEATS This routine should be used only as a debugging aid, when the condition of the semaphore is known.

RETURNS OK, or ERROR if the semaphore ID is invalid.

SEE ALSO `semMLib`, `semGive()`

semPxLibInit()

| | |
|--------------------|---|
| NAME | <i>semPxLibInit()</i> – initialize POSIX semaphore support |
| SYNOPSIS | <code>STATUS semPxLibInit (void)</code> |
| DESCRIPTION | This routine must be called before using POSIX semaphores. |
| RETURNS | OK, or ERROR if there is an error installing the semaphore library. |
| SEE ALSO | <code>semPxLib</code> |

semPxShowInit()

| | |
|--------------------|--|
| NAME | <i>semPxShowInit()</i> – initialize the POSIX semaphore show facility |
| SYNOPSIS | <code>STATUS semPxShowInit (void)</code> |
| DESCRIPTION | <p>This routine links the POSIX semaphore show routine into the VxWorks system. It is called automatically when the this show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none">– If you use the configuration header files, define <code>INCLUDE_SHOW_ROUTINES</code> in <code>config.h</code>.– If you use the Tornado project facility, select <code>INCLUDE_POSIX_SEM_SHOW</code>. |
| RETURNS | OK, or ERROR if an error occurs installing the file pointer show routine. |
| SEE ALSO | <code>semPxShow</code> |

semShow()

NAME *semShow()* – show information about a semaphore

SYNOPSIS

```
STATUS semShow
(
    SEM_ID semId, /* semaphore to display */
    int    level /* 0 = summary, 1 = details */
)
```

DESCRIPTION This routine displays the state and optionally the pended tasks of a semaphore. A summary of the state of the semaphore is displayed as follows:

```
Semaphore Id      : 0x585f2
Semaphore Type    : BINARY
Task Queuing      : PRIORITY
Pended Tasks      : 1
State              : EMPTY {Count if COUNTING, Owner if MUTEX}
```

If *level* is 1, then more detailed information will be displayed. If tasks are blocked on the queue, they are displayed in the order in which they will unblock, as follows:

| NAME | TID | PRI | DELAY |
|----------|--------|-----|-------|
| ----- | ----- | --- | ---- |
| tExcTask | 3fd678 | 0 | 21 |
| tLogTask | 3f8ac0 | 0 | 611 |

RETURNS OK or ERROR.

SEE ALSO *semShow*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

semShowInit()

NAME *semShowInit()* – initialize the semaphore show facility

SYNOPSIS `void semShowInit (void)`

DESCRIPTION This routine links the semaphore show facility into the VxWorks system. It is called automatically when the semaphore show facility is configured into VxWorks using either of the following methods:

- If you use configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.

semTake()

– If you use the Tornado project facility, select `INCLUDE_SEM_SHOW`.

RETURNS N/A

SEE ALSO `semShow`

semTake()

NAME `semTake()` – take a semaphore

SYNOPSIS

```
STATUS semTake
(
    SEM_ID semId, /* semaphore ID to take */
    int    timeout /* timeout in ticks */
)
```

DESCRIPTION This routine performs the take operation on a specified semaphore. Depending on the type of semaphore, the state of the semaphore and the calling task may be affected. The behavior of `semTake()` is discussed fully in the library description of the specific semaphore type being used.

A timeout in ticks may be specified. If a task times out, `semTake()` will return `ERROR`. Timeouts of `WAIT_FOREVER` (-1) and `NO_WAIT` (0) indicate to wait indefinitely or not to wait at all.

When `semTake()` returns due to timeout, it sets the `errno` to `S_objLib_OBJ_TIMEOUT` (defined in `objLib.h`).

The `semTake()` routine is not callable from interrupt service routines.

RETURNS `OK`, or `ERROR` if the semaphore ID is invalid or the task timed out.

ERRNO `S_intLib_NOT_ISR_CALLABLE`, `S_objLib_OBJ_ID_ERROR`, `S_objLib_OBJ_UNAVAILABLE`

SEE ALSO `semLib`, `semBLib`, `semCLib`, `semMLib`, `semSmLib`

sem_close()

| | |
|--------------------|---|
| NAME | <i>sem_close()</i> – close a named semaphore (POSIX) |
| SYNOPSIS | <pre>int sem_close (sem_t * sem /* semaphore descriptor */)</pre> |
| DESCRIPTION | <p>This routine is called to indicate that the calling task is finished with the specified named semaphore, <i>sem</i>. Do not call this routine with an unnamed semaphore (i.e., one created by <i>sem_init()</i>); the effects are undefined. The <i>sem_close()</i> call deallocates any system resources allocated by the system for use by this task for this semaphore.</p> <p>If the semaphore has not been removed with a call to <i>sem_unlink()</i>, then <i>sem_close()</i> has no effect on the state of the semaphore. However, if the semaphore has been unlinked, the semaphore vanishes when the last task closes it.</p> |
| WARNING | Take care to avoid risking the deletion of a semaphore that another task has already locked. Applications should only close semaphores that the closing task has opened. |
| RETURNS | 0 (OK), or -1 (ERROR) if unsuccessful. |
| ERRNO | EINVAL – invalid semaphore descriptor. |
| SEE ALSO | <i>semPxBLib</i> , <i>sem_unlink()</i> , <i>sem_open()</i> , <i>sem_init()</i> |

sem_destroy()

| | |
|--------------------|---|
| NAME | <i>sem_destroy()</i> – destroy an unnamed semaphore (POSIX) |
| SYNOPSIS | <pre>int sem_destroy (sem_t * sem /* semaphore descriptor */)</pre> |
| DESCRIPTION | <p>This routine is used to destroy the unnamed semaphore indicated by <i>sem</i>.</p> <p>The <i>sem_destroy()</i> call can only destroy a semaphore created by <i>sem_init()</i>. Calling <i>sem_destroy()</i> with a named semaphore will cause a EINVAL error. Subsequent use of the <i>sem</i> semaphore will cause an EINVAL error in the calling function.</p> |

sem_getvalue()

If one or more tasks is blocked on the semaphore, the semaphore is not destroyed.

| | |
|-----------------|--|
| WARNING | Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that has already locked that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully locked. |
| RETURNS | 0 (OK), or -1 (ERROR) if unsuccessful. |
| ERRNO | EINVAL – invalid semaphore descriptor. EBUSY – one or more tasks is blocked on the semaphore. |
| SEE ALSO | semPxBLib , sem_init() |

sem_getvalue()

| | |
|--------------------|--|
| NAME | <i>sem_getvalue()</i> – get the value of a semaphore (POSIX) |
| SYNOPSIS | <pre>int sem_getvalue (sem_t * sem, /* semaphore descriptor */ int * sval /* buffer by which the value is returned */)</pre> |
| DESCRIPTION | <p>This routine updates the location referenced by the <i>sval</i> argument to have the value of the semaphore referenced by <i>sem</i> without affecting the state of the semaphore. The updated value represents an actual semaphore value that occurred at some unspecified time during the call, but may not be the actual value of the semaphore when it is returned to the calling task.</p> <p>If <i>sem</i> is locked, the value returned by <i>sem_getvalue()</i> will either be zero or a negative number whose absolute value represents the number of tasks waiting for the semaphore at some unspecified time during the call.</p> |
| RETURNS | 0 (OK), or -1 (ERROR) if unsuccessful. |
| ERRNO | EINVAL – invalid semaphore descriptor. |
| SEE ALSO | semPxBLib , sem_post() , sem_trywait() , sem_trywait() |

sem_init()

| | |
|--------------------|---|
| NAME | <i>sem_init()</i> – initialize an unnamed semaphore (POSIX) |
| SYNOPSIS | <pre>int sem_init (sem_t * sem, /* semaphore to be initialized */ int pshared, /* process sharing */ unsigned int value /* semaphore initialization value */)</pre> |
| DESCRIPTION | <p>This routine is used to initialize the unnamed semaphore <i>sem</i>. The value of the initialized semaphore is <i>value</i>. Following a successful call to <i>sem_init()</i> the semaphore may be used in subsequent calls to <i>sem_wait()</i>, <i>sem_trywait()</i>, and <i>sem_post()</i>. This semaphore remains usable until the semaphore is destroyed.</p> <p>The <i>pshared</i> parameter currently has no effect.</p> <p>Only <i>sem</i> itself may be used for synchronization.</p> |
| RETURNS | 0 (OK), or -1 (ERROR) if unsuccessful. |
| ERRNO | EINVAL – <i>value</i> exceeds SEM_VALUE_MAX . ENOSPC – unable to initialize semaphore due to resource constraints. |
| SEE ALSO | <i>semPxBLib</i> , <i>sem_wait()</i> , <i>sem_trywait()</i> , <i>sem_post()</i> |

sem_open()

| | |
|--------------------|--|
| NAME | <i>sem_open()</i> – initialize/open a named semaphore (POSIX) |
| SYNOPSIS | <pre>sem_t * sem_open (const char * name, /* semaphore name */ int oflag /* semaphore creation flags */)</pre> |
| DESCRIPTION | <p>This routine establishes a connection between a named semaphore and a task. Following a call to <i>sem_open()</i> with a semaphore name <i>name</i>, the task may reference the semaphore</p> |

sem_open()

associated with *name* using the address returned by this call. This semaphore may be used in subsequent calls to *sem_wait()*, *sem_trywait()*, and *sem_post()*. The semaphore remains usable until the semaphore is closed by a successful call to *sem_close()*.

The *oflag* argument controls whether the semaphore is created or merely accessed by the call to *sem_open()*. The following flag bits may be set in *oflag*:

O_CREAT

Use this flag to create a semaphore if it does not already exist. If **O_CREAT** is set and the semaphore already exists, **O_CREAT** has no effect except as noted below under **O_EXCL**. Otherwise, *sem_open()* creates a semaphore. **O_CREAT** requires a third and fourth argument: *mode*, which is of type *mode_t*, and *value*, which is of type unsigned int. *mode* has no effect in this implementation. The semaphore is created with an initial value of *value*. Valid initial values for semaphores must be less than or equal to **SEM_VALUE_MAX**.

O_EXCL

If **O_EXCL** and **O_CREAT** are set, *sem_open()* will fail if the semaphore name exists. If **O_EXCL** is set and **O_CREAT** is not set, the named semaphore is not created.

To determine whether a named semaphore already exists in the system, call *sem_open()* with the flags **O_CREAT** | **O_EXCL**. If the *sem_open()* call fails, the semaphore exists.

If a task makes multiple calls to *sem_open()* with the same value for *name*, then the same semaphore address is returned for each such call, provided that there have been no calls to *sem_unlink()* for this semaphore.

References to copies of the semaphore will produce undefined results.

NOTE

The current implementation has the following limitations:

- A semaphore cannot be closed with calls to *_exit()* or *exec()*.
- A semaphore cannot be implemented as a file.
- Semaphore names will not appear in the file system.

RETURNS

A pointer to *sem_t*, or -1 (ERROR) if unsuccessful.

ERRNO**EEXIST**

- **O_CREAT** | **O_EXCL** are set and the semaphore already exists.

EINVAL

- *value* exceeds **SEM_VALUE_MAX** or the semaphore name is invalid.

ENAMETOOLONG

- the semaphore name is too long.

ENOENT

- the named semaphore does not exist and **O_CREAT** is not set.

ENOSPC

- the semaphore could not be initialized due to resource constraints.

SEE ALSO

semPxBLib, *sem_unlink()*

sem_post()

| | |
|--------------------|---|
| NAME | <i>sem_post()</i> – unlock (give) a semaphore (POSIX) |
| SYNOPSIS | <pre>int sem_post (sem_t * sem /* semaphore descriptor */)</pre> |
| DESCRIPTION | <p>This routine unlocks the semaphore referenced by <i>sem</i> by performing the semaphore unlock operation on that semaphore.</p> <p>If the semaphore value resulting from the operation is positive, then no tasks were blocked waiting for the semaphore to become unlocked; the semaphore value is simply incremented.</p> <p>If the value of the semaphore resulting from this semaphore is zero, then one of the tasks blocked waiting for the semaphore will return successfully from its call to <i>sem_wait()</i>.</p> |
| NOTE | <p>The _POSIX_PRIORITY_SCHEDULING functionality is not yet supported.</p> <p>Note that the POSIX terms <i>unlock</i> and <i>post</i> correspond to the term <i>give</i> used in other VxWorks semaphore documentation.</p> |
| RETURNS | 0 (OK), or -1 (ERROR) if unsuccessful. |
| ERRNO | EINVAL – invalid semaphore descriptor. |
| SEE ALSO | <i>semPxBLib</i> , <i>sem_wait()</i> , <i>sem_trywait()</i> |

sem_trywait()

| | |
|-----------------|--|
| NAME | <i>sem_trywait()</i> – lock (take) a semaphore, returning error if unavailable (POSIX) |
| SYNOPSIS | <pre>int sem_trywait (sem_t * sem /* semaphore descriptor */)</pre> |

sem_unlink()

| | |
|--------------------|---|
| DESCRIPTION | <p>This routine locks the semaphore referenced by <i>sem</i> only if the semaphore is currently not locked; that is, if the semaphore value is currently positive. Otherwise, it does not lock the semaphore. In either case, this call returns immediately without blocking.</p> <p>Upon return, the state of the semaphore is always locked (either as a result of this call or by a previous <i>sem_wait()</i> or <i>sem_trywait()</i>). The semaphore will remain locked until <i>sem_post()</i> is executed and returns successfully.</p> <p>Deadlock detection is not implemented.</p> <p>Note that the POSIX term <i>lock</i> corresponds to the term <i>take</i> used in other VxWorks semaphore documentation.</p> |
| RETURNS | 0 (OK), or -1 (ERROR) if unsuccessful. |
| ERRNO | EAGAIN – semaphore is already locked. EINVAL – invalid semaphore descriptor. |
| SEE ALSO | <i>semPxBLib</i> , <i>sem_wait()</i> , <i>sem_post()</i> |

sem_unlink()

| | |
|--------------------|--|
| NAME | <i>sem_unlink()</i> – remove a named semaphore (POSIX) |
| SYNOPSIS | <pre>int sem_unlink (const char * name /* semaphore name */)</pre> |
| DESCRIPTION | <p>This routine removes the string <i>name</i> from the semaphore name table, and marks the corresponding semaphore for destruction. An unlinked semaphore is destroyed when the last task closes it with <i>sem_close()</i>. After a particular name is removed from the table, calls to <i>sem_open()</i> using the same name cannot connect to the same semaphore, even if other tasks are still using it. Instead, such calls refer to a new semaphore with the same name.</p> |
| RETURNS | 0 (OK), or -1 (ERROR) if unsuccessful. |
| ERRNO | ENAMETOOLONG – semaphore name too long. ENOENT – named semaphore does not exist. |
| SEE ALSO | <i>semPxBLib</i> , <i>sem_open()</i> , <i>sem_close()</i> |

sem_wait()

| | |
|--------------------|--|
| NAME | <i>sem_wait()</i> – lock (take) a semaphore, blocking if not available (POSIX) |
| SYNOPSIS | <pre>int sem_wait (sem_t * sem /* semaphore descriptor */)</pre> |
| DESCRIPTION | <p>This routine locks the semaphore referenced by <i>sem</i> by performing the semaphore lock operation on that semaphore. If the semaphore value is currently zero, the calling task will not return from the call to <i>sem_wait()</i> until it either locks the semaphore or the call is interrupted by a signal.</p> <p>On return, the state of the semaphore is locked and will remain locked until <i>sem_post()</i> is executed and returns successfully.</p> <p>Deadlock detection is not implemented.</p> <p>Note that the POSIX term <i>lock</i> corresponds to the term <i>take</i> used in other VxWorks documentation regarding semaphores.</p> |
| RETURNS | 0 (OK), or -1 (ERROR) if unsuccessful. |
| ERRNO | EINVAL – invalid semaphore descriptor, or semaphore destroyed while task waiting. |
| SEE ALSO | <i>semPxBLib</i> , <i>sem_trywait()</i> , <i>sem_post()</i> |

send()

| | |
|--------------------|---|
| NAME | <i>send()</i> – send data to a socket |
| SYNOPSIS | <pre>int send (int s, /* socket to send to */ char * buf, /* pointer to buffer to transmit */ int buflen, /* length of buffer */ int flags /* flags to underlying protocols */)</pre> |
| DESCRIPTION | This routine transmits data to a previously established connection-based (stream) socket. |

sendmsg()

The maximum length of *buf* is subject to the limits on TCP buffer size; see the discussion of `SO_SNDBUF` in the `setsockopt()` manual entry.

You may OR the following values into the *flags* parameter with this operation:

`MSG_OOB` (0x1)

Out-of-band data.

`MSG_DONTROUTE` (0x4)

Send without using routing tables.

RETURNS The number of bytes sent, or `ERROR` if the call fails.

SEE ALSO `sockLib`, `setsockopt()`, `sendmsg()`

sendmsg()

NAME `sendmsg()` – send a message to a socket

SYNOPSIS

```
int sendmsg
(
    int          sd,      /* socket to send to */
    struct msghdr * mp,  /* scatter-gather message header */
    int          flags /* flags to underlying protocols */
)
```

DESCRIPTION This routine sends a message to a datagram socket. It may be used in place of `sendto()` to decrease the overhead of reconstructing the message-header structure (`msghdr`) for each message.

For BSD 4.4 sockets a copy of the `mp>msg_iov` array will be made. This requires a cluster from the network stack system pool of size `mp>msg_iovlen * sizeof (struct iovec)` or 8 bytes.

RETURNS The number of bytes sent, or `ERROR` if the call fails.

SEE ALSO `sockLib`, `sendto()`

sendto()

NAME *sendto()* – send a message to a socket

SYNOPSIS

```
int sendto
(
    int          s,          /* socket to send data to */
    caddr_t      buf,        /* pointer to data buffer */
    int          buflen,    /* length of buffer */
    int          flags,     /* flags to underlying protocols */
    struct sockaddr * to,    /* recipient's address */
    int          tolen      /* length of to sockaddr */
)
```

DESCRIPTION This routine sends a message to the datagram socket named by *to*. The socket *s* is received by the receiver as the sending socket.

The maximum length of *buf* is subject to the limits on UDP buffer size. See the discussion of `SO_SNDBUF` in the *setsockopt()* manual entry.

You can OR the following values into the *flags* parameter with this operation:

MSG_OOB (0x1)
Out-of-band data.

MSG_DONTROUTE (0x4)
Send without using routing tables.

RETURNS The number of bytes sent, or `ERROR` if the call fails.

SEE ALSO *sockLib*, *setsockopt()*

set_new_handler()

NAME *set_new_handler()* – set `new_handler` to user-defined function (C++)

SYNOPSIS `extern void (*set_new_handler (void(* pNewNewHandler)())) ()`

DESCRIPTION This function is used to define the function that will be called when operator `new` cannot allocate memory.

The `new_handler` acts for all threads in the system; you cannot set a different handler for different tasks.

set_terminate()

| | |
|----------------------|---|
| RETURNS | A pointer to the previous value of <code>new_handler</code> . |
| INCLUDE FILES | <code>new</code> |
| SEE ALSO | <code>cplusLib</code> |

set_terminate()

| | |
|----------------------|--|
| NAME | <code>set_terminate()</code> – set terminate to user-defined function (C++) |
| SYNOPSIS | <code>extern void (*set_terminate (void(* terminate_handler)())) ()</code> |
| DESCRIPTION | <p>This function is used to define the <code>terminate_handler</code> which will be called when an uncaught exception is raised.</p> <p>The <code>terminate_handler</code> acts for all threads in the system; you cannot set a different handler for different tasks.</p> |
| RETURNS | The previous <code>terminate_handler</code> . |
| INCLUDE FILES | <code>exception</code> |
| SEE ALSO | <code>cplusLib</code> |

setbuf()

| | |
|----------------------|---|
| NAME | <code>setbuf()</code> – specify the buffering for a stream (ANSI) |
| SYNOPSIS | <pre>void setbuf (FILE * fp, /* stream to set buffering for */ char * buf /* buffer to use */)</pre> |
| DESCRIPTION | <p>Except that it returns no value, this routine is equivalent to <code>setvbuf()</code> invoked with the <code>mode _IOFBF</code> (full buffering) and <code>size BUFSIZ</code>, or (if <code>buf</code> is a null pointer), with the <code>mode _IONBF</code> (no buffering).</p> |
| INCLUDE FILES | <code>stdio.h</code> |

RETURNS N/A

SEE ALSO `ansiStdio`, `setvbuf()`

setbuffer()

NAME `setbuffer()` – specify buffering for a stream

SYNOPSIS

```
void setbuffer
(
    FILE * fp, /* stream to set buffering for */
    char * buf, /* buffer to use */
    int size /* buffer size */
)
```

DESCRIPTION This routine specifies a buffer *buf* to be used for a stream in place of the automatically allocated buffer. If *buf* is NULL, the stream is unbuffered. This routine should be called only after the stream has been associated with an open file and before any other operation is performed on the stream.

This routine is provided for compatibility with earlier VxWorks releases.

INCLUDE FILES `stdio.h`

RETURNS N/A

SEE ALSO `ansiStdio`, `setvbuf()`

sethostname()

NAME `sethostname()` – set the symbolic name of this machine

SYNOPSIS

```
int sethostname
(
    char * name, /* machine name */
    int nameLen /* length of name */
)
```

DESCRIPTION This routine sets the target machine's symbolic name, which can be used for identification.

RETURNS OK or ERROR.

SEE ALSO **hostLib**

setjmp()

NAME *setjmp()* – save the calling environment in a **jmp_buf** argument (ANSI)

SYNOPSIS

```
int setjmp
(
    jmp_buf env
)
```

DESCRIPTION This routine saves the calling environment in *env*, in order to permit a *longjmp()* call to restore that environment (thus performing a non-local goto).

Constraints on Calling Environment

The *setjmp()* routine may only be used in the following contexts:

- as the entire controlling expression of a selection or iteration statement;
- as one operand of a relational or equality operator, in the controlling expression of a selection or iteration statement;
- as the operand of a single-argument ! operator, in the controlling expression of a selection or iteration statement; or
- as a complete C statement containing nothing other than the *setjmp()* call (though the result may be cast to **void**).

RETURNS * From a direct invocation, *setjmp()* returns zero. From a call to *longjmp()*, it returns a non-zero value specified as an argument to *longjmp()*.

SEE ALSO **ansiSetjmp**, *longjmp()*

setlinebuf()

| | |
|--------------------|---|
| NAME | <i>setlinebuf()</i> – set line buffering for standard output or standard error |
| SYNOPSIS | <pre>int setlinebuf (FILE * fp /* stream - stdout or stderr */)</pre> |
| DESCRIPTION | <p>This routine changes stdout or stderr streams from block-buffered or unbuffered to line-buffered. Unlike <i>setbuf()</i>, <i>setbuffer()</i>, or <i>setvbuf()</i>, it can be used at any time the stream is active.</p> <p>A stream can be changed from unbuffered or line-buffered to fully buffered using <i>freopen()</i>. A stream can be changed from fully buffered or line-buffered to unbuffered using <i>freopen()</i> followed by <i>setbuf()</i> with a buffer argument of NULL.</p> <p>This routine is provided for compatibility with earlier VxWorks releases.</p> |
| INCLUDE | stdio.h |
| RETURNS | OK, or ERROR if <i>fp</i> is not a valid stream. |
| SEE ALSO | ansiStdio |

setlocale()

| | |
|--------------------|--|
| NAME | <i>setlocale()</i> – set the appropriate locale (ANSI) |
| SYNOPSIS | <pre>char *setlocale (int category, /* category to change */ const char * localeName /* locale name */)</pre> |
| DESCRIPTION | <p>This function is included for ANSI compatibility. Only the default is implemented. At program start-up, the equivalent of the following is executed:</p> <pre>setlocale (LC_ALL, "C");</pre> <p>This specifies the program's entire locale and the minimal environment for C translation.</p> |

setsockopt()

| | |
|----------------------|------------------------------|
| INCLUDE FILES | locale.h, string.h, stdlib.h |
| RETURNS | A pointer to the string "C". |
| SEE ALSO | ansiLocale |

setsockopt()

NAME *setsockopt()* – set socket options

SYNOPSIS

```

STATUS setsockopt
(
    int    s,          /* target socket */
    int    level,     /* protocol level of option */
    int    optname,   /* option name */
    char * optval,    /* pointer to option value */
    int    optlen     /* option length */
)

```

DESCRIPTION This routine sets the options associated with a socket. To manipulate options at the "socket" level, *level* should be `SOL_SOCKET`. Any other levels should use the appropriate protocol number.

OPTIONS FOR STREAM SOCKETS

The following sections discuss the socket options available for stream (TCP) sockets.

SO_KEEPALIVE -- Detecting a Dead Connection

Specify the `SO_KEEPALIVE` option to make the transport protocol (TCP) initiate a timer to detect a dead connection:

```
setsockopt (sock, SOL_SOCKET, SO_KEEPALIVE, &optval, sizeof (optval));
```

This prevents an application from hanging on an invalid connection. The value at *optval* for this option is an integer (type `int`), either 1 (on) or 0 (off).

The integrity of a connection is verified by transmitting zero-length TCP segments triggered by a timer, to force a response from a peer node. If the peer does not respond after repeated transmissions of the `KEEPALIVE` segments, the connection is dropped, all protocol data structures are reclaimed, and processes sleeping on the connection are awakened with an `ETIMEDOUT` error.

The `ETIMEDOUT` timeout can happen in two ways. If the connection is not yet established, the `KEEPALIVE` timer expires after idling for `TCPTV_KEEP_INIT`. If the connection is established, the `KEEPALIVE` timer starts up when there is no traffic for

TCPTV_KEEP_IDLE. If no response is received from the peer after sending the KEEPALIVE segment TCPTV_KEEPCNT times with interval TCPTV_KEEPINTVL, TCP assumes that the connection is invalid. The parameters TCPTV_KEEP_INIT, TCPTV_KEEP_IDLE, TCPTV_KEEPCNT, and TCPTV_KEEPINTVL are defined in the file `target/h/net/tcp_timer.h`.

SO_LINGER -- Closing a Connection

Specify the `SO_LINGER` option to determine whether TCP should perform a "graceful" close:

```
setsockopt (sock, SOL_SOCKET, SO_LINGER, &optval, sizeof (optval));
```

For a "graceful" close in response to the shutdown of a connection, TCP tries to make sure that all the unacknowledged data in transmission channel are acknowledged, and the peer is shut down properly, by going through an elaborate set of state transitions.

The value at *optval* indicates the amount of time to linger if there is unacknowledged data, using `struct linger` in `target/h/sys/socket.h`. The `linger` structure has two members: `l_onoff` and `l_linger`. `l_onoff` can be set to 1 to turn on the `SO_LINGER` option, or set to 0 to turn off the `SO_LINGER` option. `l_linger` indicates the amount of time to linger. If `l_onoff` is turned on and `l_linger` is set to 0, a default value `TCP_LINGERTIME` (specified in `netinet/tcp_timer.h`) is used for incoming connections accepted on the socket.

When `SO_LINGER` is turned on and the `l_linger` field is set to 0, TCP simply drops the connection by sending out an RST if a connection is already established; frees up the space for the TCP protocol control block; and wakes up all tasks sleeping on the socket.

For the client side socket, the value of `l_linger` is not changed if it is set to 0. To make sure that the value of `l_linger` is 0 on a newly accepted socket connection, issue another `setsockopt()` after the `accept()` call.

Currently the exact value of `l_linger` time is actually ignored (other than checking for 0); that is, TCP performs the state transitions if `l_linger` is not 0, but does not explicitly use its value.

TCP_NODELAY -- Delivering Messages Immediately

Specify the `TCP_NODELAY` option for real-time protocols, such as the X Window System Protocol, that require immediate delivery of many small messages:

```
setsockopt (sock, IPPROTO_TCP, TCP_NODELAY, &optval, sizeof (optval));
```

The value at *optval* is an integer (type `int`) set to either 1 (on) or 0 (off).

By default, the VxWorks TCP implementation employs an algorithm that attempts to avoid the congestion that can be produced by a large number of small TCP segments. This typically arises with virtual terminal applications (such as telnet or rlogin) across networks that have low bandwidth and long delays. The algorithm attempts to have no more than one outstanding unacknowledged segment in the transmission channel while queueing up the rest of the smaller segments for later transmission. Another segment is

setsockopt()

sent only if enough new data is available to make up a maximum sized segment, or if the outstanding data is acknowledged.

This congestion-avoidance algorithm works well for virtual terminal protocols and bulk data transfer protocols such as FTP without any noticeable side effects. However, real-time protocols that require immediate delivery of many small messages, such as the X Window System Protocol, need to defeat this facility to guarantee proper responsiveness in their operation.

TCP_NODELAY is a mechanism to turn off the use of this algorithm. If this option is turned on and there is data to be sent out, TCP bypasses the congestion-avoidance algorithm: any available data segments are sent out if there is enough space in the send window.

SO_DEBUG -- Debugging the underlying protocol

Specify the SO_DEBUG option to let the underlying protocol module record debug information.

```
setsockopt (sock, SOL_SOCKET, SO_KEEPAALIVE, &optval, sizeof (optval));
```

The value at *optval* for this option is an integer (type **int**), either 1 (on) or 0 (off).

OPTION FOR DATAGRAM SOCKETS

The following section discusses an option for datagram (UDP) sockets.

SO_BROADCAST -- Sending to Multiple Destinations

Specify the SO_BROADCAST option when an application needs to send data to more than one destination:

```
setsockopt (sock, SOL_SOCKET, SO_BROADCAST, &optval, sizeof (optval));
```

The value at *optval* is an integer (type *int*), either 1 (on) or 0 (off).

OPTIONS FOR DATAGRAM AND RAW SOCKETS

The following section discusses options for multicasting on UDP and RAW sockets.

IP_ADD_MEMBERSHIP -- Join a Multicast Group

Specify the IP_ADD_MEMBERSHIP option when a process needs to join multicast group:

```
setsockopt (sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *)&ipMreq,
           sizeof (ipMreq));
```

The value of *ipMreq* is an **ip_mreq** structure. **ipMreq.imr_multiaddr.s_addr** is the internet multicast address **ipMreq.imr_interface.s_addr** is the internet unicast address of the interface through which the multicast packet needs to pass.

IP_DROP_MEMBERSHIP -- Leave a Multicast Group

Specify the IP_DROP_MEMBERSHIP option when a process needs to leave a previously joined multicast group:

```
setsockopt (sock, IPPROTO_IP, IP_DROP_MEMBERSHIP, (char *)&ipMreq,
           sizeof (ipMreq));
```

The value of *ipMreq* is an `ip_mreq` structure. `ipMreq.imr_multiaddr.s_addr` is the internet multicast address. `ipMreq.imr_interface.s_addr` is the internet unicast address of the interface to which the multicast address was bound.

IP_MULTICAST_IF -- Select a Default Interface for Outgoing Multicasts

Specify the `IP_MULTICAST_IF` option when an application needs to specify an outgoing network interface through which all multicast packets are sent:

```
setsockopt (sock, IPPROTO_IP, IP_MULTICAST_IF, (char *)&ifAddr,
           sizeof (mCastAddr));
```

The value of *ifAddr* is an `in_addr` structure. `ifAddr.s_addr` is the internet network interface address.

IP_MULTICAST_TTL -- Select a Default TTL

Specify the `IP_MULTICAST_TTL` option when an application needs to select a default TTL (time to live) for outgoing multicast packets:

```
setsockopt (sock, IPPROTO_IP, IP_MULTICAST_TTL, &optval, sizeof(optval));
```

The value at *optval* is an integer (type *int*), time to live value.

| optval(TTL) | Application | Scope |
|--------------------|-------------------------------|----------------|
| 0 | | same interface |
| 1 | | same subnet |
| 31 | local event video | |
| 32 | | same site |
| 63 | local event audio | |
| 64 | | same region |
| 95 | IETF channel 2 video | |
| 127 | IETF channel 1 video | |
| 128 | | same continent |
| 159 | IETF channel 2 audio | |
| 191 | IETF channel 1 audio | |
| 223 | IETF channel 2 low-rate audio | |
| 255 | IETF channel 1 low-rate audio | |
| | unrestricted in scope | |

IP_MULTICAST_LOOP -- Enable or Disable Loopback

Enable or disable loopback of outgoing multicasts.

```
setsockopt (sock, IPPROTO_IP, IP_MULTICAST_LOOP, &optval, sizeof(optval));
```

The value at *optval* is an integer (type *int*), either 1(on) or 0 (off).

setsockopt()**OPTIONS FOR BOTH STREAM AND DATAGRAM SOCKETS**

The following options can be used with either stream or datagram sockets.

SO_REUSEADDR -- Reusing a Socket Address

Specify **SO_REUSEADDR** to bind a stream socket to a local port that may be still bound to another stream socket:

```
setsockopt (sock, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof (optval));
```

The value at *optval* is an integer (type *int*), either 1 (on) or 0 (off).

When the **SO_REUSEADDR** option is turned on, applications may bind a stream socket to a local port even if it is still bound to another stream socket, if that other socket is associated with a "zombie" protocol control block context not yet freed from previous sessions. The uniqueness of port number combinations for each connection is still preserved through sanity checks performed at actual connection setup time. If this option is not turned on and an application attempts to bind to a port which is being used by a zombie protocol control block, the *bind()* call fails.

SO_SNDBUF -- Specifying the Size of the Send Buffer

Specify **SO_SNDBUF** to adjust the maximum size of the socket-level send buffer:

```
setsockopt (sock, SOL_SOCKET, SO_SNDBUF, &optval, sizeof (optval));
```

The value at *optval* is an integer (type **int**) that specifies the size of the socket-level send buffer to be allocated.

When stream or datagram sockets are created, each transport protocol reserves a set amount of space at the socket level for use when the sockets are attached to a protocol. For TCP, the default size of the send buffer is 8192 bytes. For UDP, the default size is 9216 bytes. Socket-level buffers are allocated dynamically from the mbuf pool.

The effect of setting the maximum size of buffers (for both **SO_SNDBUF** and **SO_RCVBUF**, described below) is not actually to allocate the mbufs from the mbuf pool, but to set the high-water mark in the protocol data structure which is used later to limit the amount of mbuf allocation. Thus, the maximum size specified for the socket level send and receive buffers can affect the performance of bulk data transfers. For example, the size of the TCP receive windows is limited by the remaining socket-level buffer space. These parameters must be adjusted to produce the optimal result for a given application.

SO_RCVBUF -- Specifying the Size of the Receive Buffer

Specify **SO_RCVBUF** to adjust the maximum size of the socket-level receive buffer:

```
setsockopt (sock, SOL_SOCKET, SO_RCVBUF, &optval, sizeof (optval));
```

The value at *optval* is an integer (type **int**) that specifies the size of the socket-level receive buffer to be allocated.

When stream or datagram sockets are created, each transport protocol reserves a set amount of space at the socket level for use when the sockets are attached to a protocol.

For TCP, the default size is 8192 bytes. UDP reserves 41600 bytes, enough space for up to forty incoming datagrams (1 Kbyte each).

See the `SO_SNDBUF` discussion above for a discussion of the impact of buffer size on application performance.

SO_OOBINLINE -- Placing Urgent Data in the Normal Data Stream

Specify the `SO_OOBINLINE` option to place urgent data within the normal receive data stream:

```
setsockopt (sock, SOL_SOCKET, SO_OOBINLINE, &optval, sizeof (optval));
```

TCP provides an expedited data service which does not conform to the normal constraints of sequencing and flow control of data streams. The expedited service delivers "out-of-band" (urgent) data ahead of other "normal" data to provide interrupt-like services (for example, when you hit a CTRL-C during telnet or rlogin session while data is being displayed on the screen.)

TCP does not actually maintain a separate stream to support the urgent data. Instead, urgent data delivery is implemented as a pointer (in the TCP header) which points to the sequence number of the octet following the urgent data. If more than one transmission of urgent data is received from the peer, they are all put into the normal stream. This is intended for applications that cannot afford to miss out on any urgent data but are usually too slow to respond to them promptly.

RETURNS OK, or ERROR if there is an invalid socket, an unknown option, an option length greater than MLEN, insufficient mbufs, or the call is unable to set the specified option.

SEE ALSO `sockLib`

setvbuf()

NAME `setvbuf()` – specify buffering for a stream (ANSI)

SYNOPSIS

```
int setvbuf
(
    FILE * fp, /* stream to set buffering for */
    char * buf, /* buffer to use (optional) */
    int mode, /* _IOFBF = fully buffered */
              /* _IOLBF = line buffered */
              /* _IONBF = unbuffered */
    size_t size /* buffer size */
)
```

shell()

DESCRIPTION This routine sets the buffer size and buffering mode for a specified stream. It should be called only after the stream has been associated with an open file and before any other operation is performed on the stream. The argument *mode* determines how the stream will be buffered, as follows:

`_IOFBF`
input/output is to be fully buffered.

`_IOLBF`
input/output is to be line buffered.

`_IONBF`
input/output is to be unbuffered.

If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by *setvbuf()*. The argument *size* specifies the size of the array. The contents of the array at any time are indeterminate.

INCLUDE FILES `stdio.h`

RETURNS Zero, or non-zero if *mode* is invalid or the request cannot be honored.

SEE ALSO `ansiStdio`

shell()

NAME `shell()` – the shell entry point

SYNOPSIS

```
void shell
(
    BOOL interactive /* should be TRUE, except for a script */
)
```

DESCRIPTION This routine is the shell task. It is called with a single parameter indicating whether this is an interactive shell to be used from a terminal or a socket, or a shell that executes a script.

Normally, the shell is spawned in interactive mode by the root task, `usrRoot()`, when VxWorks starts up. After that, `shell()` is called only to execute scripts, or when the shell is restarted after an abort.

The shell gets its input from standard input and sends output to standard output. Both standard input and standard output are initially assigned to the console, but are redirected by `telnetdTask()` and `rlogindTask()`.

The shell is not reentrant, since `yacc` does not generate a reentrant parser. Therefore, there can be only a single shell executing at one time.

RETURNS N/A

SEE ALSO **shellLib**, *VxWorks Programmer's Guide: Target Shell*

shellHistory()

NAME *shellHistory()* – display or set the size of shell history

SYNOPSIS

```
void shellHistory
(
    int size /* 0 = display, >0 = set history to new size */
)
```

DESCRIPTION This routine displays shell history, or resets the default number of commands displayed by shell history to *size*. By default, history size is 20 commands. Shell history is actually maintained by **ledLib**.

RETURNS N/A

SEE ALSO **shellLib**, **ledLib**, **h()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

shellInit()

NAME *shellInit()* – start the shell

SYNOPSIS

```
STATUS shellInit
(
    int stackSize, /* shell stack (0 = previous/default value) */
    int arg        /* argument to shell task */
)
```

DESCRIPTION This routine starts the shell task. If the configuration macro **INCLUDE_SHELL** is defined, *shellInit()* is called by the root task, *usrRoot()*, in **usrConfig.c**.

RETURNS OK or ERROR.

SEE ALSO **shellLib**, *VxWorks Programmer's Guide: Target Shell*

shellLock()

| | |
|--------------------|--|
| NAME | <i>shellLock()</i> – lock access to the shell |
| SYNOPSIS | <pre>BOOL shellLock (BOOL request /* TRUE = lock, FALSE = unlock */)</pre> |
| DESCRIPTION | This routine locks or unlocks access to the shell. When locked, cooperating tasks, such as <i>telnetdTask()</i> and <i>rlogindTask()</i> , will not take the shell. |
| RETURNS | TRUE if <i>request</i> is "lock" and the routine successfully locks the shell, otherwise FALSE. TRUE if <i>request</i> is "unlock" and the routine successfully unlocks the shell, otherwise FALSE. |
| SEE ALSO | <i>shellLib</i> , <i>VxWorks Programmer's Guide: Target Shell</i> |

shellOrigStdSet()

| | |
|--------------------|--|
| NAME | <i>shellOrigStdSet()</i> – set the shell's default input/output/error file descriptors |
| SYNOPSIS | <pre>void shellOrigStdSet (int which, /* STD_IN, STD_OUT, STD_ERR */ int fd /* fd to be default */)</pre> |
| DESCRIPTION | This routine is called to change the shell's default standard input/output/error file descriptor. Normally, it is used only by the shell, <i>rlogindTask()</i> , and <i>telnetdTask()</i> . Values for <i>which</i> can be <i>STD_IN</i> , <i>STD_OUT</i> , or <i>STD_ERR</i> , as defined in <i>vxWorks.h</i> . Values for <i>fd</i> can be the file descriptor for any file or device. |
| RETURNS | N/A |
| SEE ALSO | <i>shellLib</i> |

shellPromptSet()

| | |
|--------------------|--|
| NAME | <i>shellPromptSet()</i> – change the shell prompt |
| SYNOPSIS | <pre>void shellPromptSet (char * newPrompt /* string to become new shell prompt */)</pre> |
| DESCRIPTION | This routine changes the shell prompt string to <i>newPrompt</i> . |
| RETURNS | N/A |
| SEE ALSO | shellLib , <i>VxWorks Programmer's Guide: Target Shell</i> , windsh , <i>Tornado User's Guide: Shell</i> |

shellScriptAbort()

| | |
|--------------------|--|
| NAME | <i>shellScriptAbort()</i> – signal the shell to stop processing a script |
| SYNOPSIS | <pre>void shellScriptAbort (void)</pre> |
| DESCRIPTION | This routine signals the shell to abort processing a script file. It can be called from within a script if an error is detected. |
| RETURNS | N/A |
| SEE ALSO | shellLib , <i>VxWorks Programmer's Guide: Target Shell</i> |

show()

| | |
|-----------------|---|
| NAME | <i>show()</i> – print information on a specified object |
| SYNOPSIS | <pre>void show (int objId, /* object ID */ int level /* information level */)</pre> |

shutdown()

| | |
|--------------------|---|
| DESCRIPTION | This command prints information on the specified object. System objects include tasks, local and shared semaphores, local and shared message queues, local and shared memory partitions, watchdogs, and symbol tables. An information level is interpreted by the objects show routine on a class by class basis. Refer to the object's library manual page for more information. |
| RETURNS | N/A |
| SEE ALSO | usrLib , i() , ti() , lkup() , <i>VxWorks Programmer's Guide: Target Shell</i> , windsh , <i>Tornado User's Guide: Shell</i> |

shutdown()

| | |
|--------------------|---|
| NAME | shutdown() – shut down a network connection |
| SYNOPSIS | <pre> STATUS shutdown (int s, /* socket to shut down */ int how /* 0 = receives disallowed */ /* 1 = sends disallowed */ /* 2 = sends and receives disallowed */) </pre> |
| DESCRIPTION | This routine shuts down all, or part, of a connection-based socket <i>s</i> . If the value of <i>how</i> is 0, receives are disallowed. If <i>how</i> is 1, sends are disallowed. If <i>how</i> is 2, both sends and receives are disallowed. |
| RETURNS | OK, or ERROR if the socket is invalid or not connected. |
| SEE ALSO | sockLib |

sigaction()

| | |
|-----------------|---|
| NAME | sigaction() – examine and/or specify the action associated with a signal (POSIX) |
| SYNOPSIS | <pre> int sigaction (int signo, /* signal of handler of interest */ const struct sigaction * pAct, /* location of new handler */ </pre> |

```

        struct sigaction *      pOact /* location to store old handler */
    )

```

| | |
|--------------------|---|
| DESCRIPTION | This routine allows the calling process to examine and/or specify the action to be associated with a specific signal. |
| RETURNS | OK (0), or ERROR (-1) if the signal number is invalid. |
| ERRNO | EINVAL |
| SEE ALSO | sigLib |

sigaddset()

NAME *sigaddset()* – add a signal to a signal set (POSIX)

SYNOPSIS

```

int sigaddset
(
    sigset_t * pSet, /* signal set to add signal to */
    int      signo /* signal to add */
)

```

DESCRIPTION This routine adds the signal specified by *signo* to the signal set specified by *pSet*.

RETURNS OK (0), or ERROR (-1) if the signal number is invalid.

ERRNO EINVAL

SEE ALSO sigLib

sigblock()

NAME *sigblock()* – add to a set of blocked signals

SYNOPSIS

```

int sigblock
(
    int mask /* mask of additional signals to be blocked */
)

```

- DESCRIPTION** This routine adds the signals in *mask* to the task's set of blocked signals. A one (1) in the bit mask indicates that the specified signal is blocked from delivery. Use the macro **SIGMASK** to construct the mask for a specified signal number.
- RETURNS** The previous value of the signal mask.
- SEE ALSO** *sigLib*, *sigprocmask()*

sigdelset()

- NAME** *sigdelset()* – delete a signal from a signal set (POSIX)
- SYNOPSIS**
- ```
int sigdelset
(
 sigset_t * pSet, /* signal set to delete signal from */
 int signo /* signal to delete */
)
```
- DESCRIPTION** This routine deletes the signal specified by *signo* from the signal set specified by *pSet*.
- RETURNS** OK (0), or ERROR (-1) if the signal number is invalid.
- ERRNO** EINVAL
- SEE ALSO** *sigLib*

---

## ***sigemptyset()***

- NAME** *sigemptyset()* – initialize a signal set with no signals included (POSIX)
- SYNOPSIS**
- ```
int sigemptyset
(
    sigset_t * pSet /* signal set to initialize */
)
```
- DESCRIPTION** This routine initializes the signal set specified by *pSet*, such that all signals are excluded.
- RETURNS** OK (0), or ERROR (-1) if the signal set cannot be initialized.

ERRNO No errors are detectable.

SEE ALSO sigLib

sigfillset()

NAME *sigfillset()* – initialize a signal set with all signals included (POSIX)

SYNOPSIS

```
int sigfillset
(
    sigset_t * pSet /* signal set to initialize */
)
```

DESCRIPTION This routine initializes the signal set specified by *pSet*, such that all signals are included.

RETURNS OK (0), or ERROR (-1) if the signal set cannot be initialized.

ERRNO No errors are detectable.

SEE ALSO sigLib

sigInit()

NAME *sigInit()* – initialize the signal facilities

SYNOPSIS

```
int sigInit (void)
```

DESCRIPTION This routine initializes the signal facilities. It is usually called from the system start-up routine *usrInit()* in *usrConfig*, before interrupts are enabled.

RETURNS OK, or ERROR if the delete hooks cannot be installed.

ERRNO S_taskLib_TASK_HOOK_TABLE_FULL

SEE ALSO sigLib

sigismember()

sigismember()

| | |
|--------------------|---|
| NAME | <i>sigismember()</i> – test to see if a signal is in a signal set (POSIX) |
| SYNOPSIS | <pre>int sigismember (const sigset_t * pSet, /* signal set to test */ int signo /* signal to test for */)</pre> |
| DESCRIPTION | This routine tests whether the signal specified by <i>signo</i> is a member of the set specified by <i>pSet</i> . |
| RETURNS | 1 if the specified signal is a member of the specified set, OK (0) if it is not, or ERROR (-1) if the test fails. |
| ERRNO | EINVAL |
| SEE ALSO | sigLib |

signal()

| | |
|--------------------|---|
| NAME | <i>signal()</i> – specify the handler associated with a signal |
| SYNOPSIS | <pre>void (*signal (int signo, void (*pHandler) ())) ()</pre> |
| DESCRIPTION | This routine chooses one of three ways in which receipt of the signal number <i>signo</i> is to be subsequently handled. If the value of <i>pHandler</i> is SIG_DFL , default handling for that signal will occur. If the value of <i>pHandler</i> is SIG_IGN , the signal will be ignored. Otherwise, <i>pHandler</i> must point to a function to be called when that signal occurs. |
| RETURNS | The value of the previous signal handler, or SIG_ERR . |
| SEE ALSO | sigLib |

sigpending()

| | |
|--------------------|---|
| NAME | <i>sigpending()</i> – retrieve the set of pending signals blocked from delivery (POSIX) |
| SYNOPSIS | <pre>int sigpending (sigset_t * pSet /* location to store pending signal set */)</pre> |
| DESCRIPTION | This routine stores the set of signals that are blocked from delivery and that are pending for the calling process in the space pointed to by <i>pSet</i> . |
| RETURNS | OK (0), or ERROR (-1) if the signal TCB cannot be allocated. |
| ERRNO | ENOMEM |
| SEE ALSO | sigLib |

sigprocmask()

| | |
|--------------------|--|
| NAME | <i>sigprocmask()</i> – examine and/or change the signal mask (POSIX) |
| SYNOPSIS | <pre>int sigprocmask (int how, /* how signal mask will be changed */ const sigset_t * pSet, /* location of new signal mask */ sigset_t * pOset /* location to store old signal mask */)</pre> |
| DESCRIPTION | <p>This routine allows the calling process to examine and/or change its signal mask. If the value of <i>pSet</i> is not NULL, it points to a set of signals to be used to change the currently blocked set.</p> <p>The value of <i>how</i> indicates the manner in which the set is changed and consists of one of the following, defined in signal.h:</p> <p>SIG_BLOCK the resulting set is the union of the current set and the signal set pointed to by <i>pSet</i>.</p> <p>SIG_UNBLOCK the resulting set is the intersection of the current set and the complement of the signal set pointed to by <i>pSet</i>.</p> |

SIG_SETMASK

the resulting set is the signal set pointed to by *pSset*.

RETURNS OK (0), or ERROR (-1) if *how* is invalid.

ERRNO EINVAL

SEE ALSO *sigLib*, *sigsetmask()*, *sigblock()*

sigqueue()

NAME *sigqueue()* – send a queued signal to a task

SYNOPSIS

```
int sigqueue
(
    int          tid,
    int          signo,
    const union sigval value
)
```

DESCRIPTION The function *sigqueue()* sends the signal specified by *signo* with the signal-parameter value specified by *value* to the process specified by *tid*.

RETURNS OK (0), or ERROR (-1) if the task ID or signal number is invalid, or if there are no queued-signal buffers available.

ERRNO EINVAL EAGAIN

SEE ALSO *sigLib*

sigqueueInit()

NAME *sigqueueInit()* – initialize the queued signal facilities

SYNOPSIS

```
int sigqueueInit
(
    int nQueues
)
```


| | |
|--------------------|---|
| DESCRIPTION | <p>This routine initializes the queued signal facilities. It must be called before any call to <i>sigqueue()</i>. It is usually called from the system start-up routine <i>usrInit()</i> in <i>usrConfig</i>, after <i>sysInit()</i> is called.</p> <p>It allocates <i>nQueues</i> buffers to be used by <i>sigqueue()</i>. A buffer is used by each call to <i>sigqueue()</i> and freed when the signal is delivered (thus if a signal is block, the buffer is unavailable until the signal is unblocked.)</p> |
| RETURNS | OK, or ERROR if memory could not be allocated. |
| SEE ALSO | <i>sigLib</i> |

sigsetmask()

| | |
|--------------------|---|
| NAME | <i>sigsetmask()</i> – set the signal mask |
| SYNOPSIS | <pre>int sigsetmask (int mask /* new signal mask */)</pre> |
| DESCRIPTION | <p>This routine sets the calling task's signal mask to a specified value. A one (1) in the bit mask indicates that the specified signal is blocked from delivery. Use the macro SIGMASK to construct the mask for a specified signal number.</p> |
| RETURNS | The previous value of the signal mask. |
| SEE ALSO | <i>sigLib</i> , <i>sigprocmask()</i> |

sigsuspend()

| | |
|-----------------|---|
| NAME | <i>sigsuspend()</i> – suspend the task until delivery of a signal (POSIX) |
| SYNOPSIS | <pre>int sigsuspend (const sigset_t * pset /* signal mask while suspended */)</pre> |

sigtimedwait()

| | |
|--------------------|--|
| DESCRIPTION | This routine suspends the task until delivery of a signal. While suspended, <i>pSet</i> is used as the set of masked signals. |
| NOTE | Since the <i>sigsuspend()</i> function suspends thread execution indefinitely, there is no successful completion return value. |
| RETURNS | -1, always. |
| ERRNO | EINTR |
| SEE ALSO | sigLib |

sigtimedwait()

NAME *sigtimedwait()* – wait for a signal

SYNOPSIS

```
int sigtimedwait
(
    const sigset_t *    pSet,    /* the signal mask while suspended */
    struct siginfo *   pInfo,    /* return value */
    const struct timespec * pTimeout
)
```

DESCRIPTION The function *sigtimedwait()* selects the pending signal from the set specified by *pSet*. If multiple signals in *pSet* are pending, it will remove and return the lowest numbered one. If no signal in *pSet* is pending at the time of the call, the task will be suspend until one of the signals in *pSet* become pending, it is interrupted by an unblocked caught signal, or until the time interval specified by *pTimeout* has expired. If *pTimeout* is NULL, then the timeout interval is forever.

If the *pInfo* argument is non-NULL, the selected signal number is stored in the **si_signo** member, and the cause of the signal is stored in the **si_code** member. If the signal is a queued signal, the value is stored in the **si_value** member of *pInfo*; otherwise the content of **si_value** is undefined.

The following values are defined in **signal.h** for **si_code**:

SI_USER
the signal was sent by the *kill()* function.

SI_QUEUE
the signal was sent by the *sigqueue()* function.

SI_TIMER
the signal was generated by the expiration of a timer set by *timer_settime()*.

SI_ASYNCIO

the signal was generated by the completion of an asynchronous I/O request.

SI_MESGQ

the signal was generated by the arrival of a message on an empty message queue.

The function *sigtimedwait()* provides a synchronous mechanism for tasks to wait for asynchronously generated signals. A task should use *sigprocmask()* to block any signals it wants to handle synchronously and leave their signal handlers in the default state. The task can then make repeated calls to *sigtimedwait()* to remove any signals that are sent to it.

RETURNS

Upon successful completion (that is, one of the signals specified by *pSet* is pending or is generated) *sigtimedwait()* will return the selected signal number. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

ERRNO**EINTR**

The wait was interrupted by an unblocked, caught signal.

EAGAIN

No signal specified by *pSet* was delivered within the specified timeout period.

EINVAL

The *pTimeout* argument specified a **tv_nsec** value less than zero or greater than or equal to 1000 million.

SEE ALSO

sigLib

sigvec()

NAME

sigvec() – install a signal handler

SYNOPSIS

```
int sigvec
(
    int                sig, /* signal to attach handler to */
    const struct sigvec * pVec, /* new handler information */
    struct sigvec *     pOvec /* previous handler information */
)
```

DESCRIPTION

This routine binds a signal handler routine referenced by *pVec* to a specified signal *sig*. It can also be used to determine which handler, if any, has been bound to a particular signal: *sigvec()* copies current signal handler information for *sig* to *pOvec* and does not install a signal handler if *pVec* is set to NULL (0).

sigwaitinfo()

Both *pVec* and *pOvec* are pointers to a structure of type **struct sigvec**. The information passed includes not only the signal handler routine, but also the signal mask and additional option bits. The structure **sigvec** and the available options are defined in **signal.h**.

| | |
|-----------------|--|
| RETURNS | OK (0), or ERROR (-1) if the signal number is invalid or the signal TCB cannot be allocated. |
| ERRNO | EINVAL, ENOMEM |
| SEE ALSO | sigLib |

sigwaitinfo()

| | |
|--------------------|---|
| NAME | <i>sigwaitinfo()</i> – wait for real-time signals |
| SYNOPSIS | <pre>int sigwaitinfo (const sigset_t * pSet, /* the signal mask while suspended */ struct siginfo * pInfo /* return value */)</pre> |
| DESCRIPTION | The function <i>sigwaitinfo()</i> is equivalent to calling <i>sigtimedwait()</i> with <i>pTimeout</i> equal to NULL. See that manual entry for more information. |
| RETURNS | Upon successful completion (that is, one of the signals specified by <i>pSet</i> is pending or is generated) <i>sigwaitinfo()</i> returns the selected signal number. Otherwise, a value of -1 is returned and errno is set to indicate the error. |
| ERRNO | EINTR The wait was interrupted by an unblocked, caught signal. |
| SEE ALSO | sigLib |

sin()

| | |
|----------------------|---|
| NAME | <i>sin()</i> – compute a sine (ANSI) |
| SYNOPSIS | <pre>double sin (double x /* angle in radians */)</pre> |
| DESCRIPTION | This routine computes the sine of <i>x</i> in double precision. The angle <i>x</i> is expressed in radians. |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision sine of <i>x</i> . |
| SEE ALSO | ansiMath, mathALib |

sincos()

| | |
|----------------------|---|
| NAME | <i>sincos()</i> – compute both a sine and cosine |
| SYNOPSIS | <pre>void sincos (double x, /* angle in radians */ double * sinResult, /* sine result buffer */ double * cosResult /* cosine result buffer */)</pre> |
| DESCRIPTION | This routine computes both the sine and cosine of <i>x</i> in double precision. The sine is copied to <i>sinResult</i> and the cosine is copied to <i>cosResult</i> . |
| INCLUDE FILES | math.h |
| RETURNS | N/A |
| SEE ALSO | mathALib |

sincosf()

sincosf()

NAME *sincosf()* – compute both a sine and cosine

SYNOPSIS

```
void sincosf
(
    float x,          /* angle in radians */
    float * sinResult, /* sine result buffer */
    float * cosResult /* cosine result buffer */
)
```

DESCRIPTION This routine computes both the sine and cosine of x in single precision. The sine is copied to *sinResult* and the cosine is copied to *cosResult*. The angle x is expressed in radians.

INCLUDE FILES **math.h**

RETURNS N/A

SEE ALSO **mathALib**

sinf()

NAME *sinf()* – compute a sine (ANSI)

SYNOPSIS

```
float sinf
(
    float x /* angle in radians */
)
```

DESCRIPTION This routine returns the sine of x in single precision. The angle x is expressed in radians.

INCLUDE FILES **math.h**

RETURNS The single-precision sine of x .

SEE ALSO **mathALib**

sinh()

| | |
|----------------------|---|
| NAME | <i>sinh()</i> – compute a hyperbolic sine (ANSI) |
| SYNOPSIS | <pre>double sinh (double x /* number whose hyperbolic sine is required */)</pre> |
| DESCRIPTION | This routine returns the hyperbolic sine of x in double precision (IEEE double, 53 bits). A range error occurs if x is too large. |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision hyperbolic sine of x . Special cases: If x is +INF, -INF, or NaN, <i>sinh()</i> returns x . |
| SEE ALSO | ansiMath, mathALib |

sinhf()

| | |
|----------------------|---|
| NAME | <i>sinhf()</i> – compute a hyperbolic sine (ANSI) |
| SYNOPSIS | <pre>float sinhF (float x /* number whose hyperbolic sine is required */)</pre> |
| DESCRIPTION | This routine returns the hyperbolic sine of x in single precision. |
| INCLUDE FILES | math.h |
| RETURNS | The single-precision hyperbolic sine of x . |
| SEE ALSO | mathALib |

slattach()

NAME *slattach()* – publish the **sl** network interface and initialize the driver and device

SYNOPSIS

```
STATUS slattach
(
    int unit,          /* SLIP device unit number */
    int fd,           /* fd of tty device for SLIP interface */
    BOOL compressEnable, /* explicitly enable CSLIP compression */
    BOOL compressAllow, /* enable CSLIP compression on Rx */
    int mtu           /* user settable MTU */
)
```

DESCRIPTION This routine publishes the **sl** interface by filling in a network interface record and adding this record to the system list. It also initializes the driver and the device to the operational state.

This routine is usually called by *slipInit()*.

RETURNS OK or ERROR.

SEE ALSO *if_sl*

slipBaudSet()

NAME *slipBaudSet()* – set the baud rate for a SLIP interface

SYNOPSIS

```
STATUS slipBaudSet
(
    int unit, /* SLIP device unit number */
    int baud /* baud rate */
)
```

DESCRIPTION This routine adjusts the baud rate of a tty device attached to a SLIP interface. It provides a way to modify the baud rate of a tty device being used as a SLIP interface.

RETURNS OK, or ERROR if the unit number is invalid or uninitialized.

SEE ALSO *if_sl*

slipDelete()

NAME *slipDelete()* – delete a SLIP interface

SYNOPSIS

```
STATUS slipDelete
(
    int unit /* SLIP unit number */
)
```

DESCRIPTION This routine resets a specified SLIP interface. It detaches the tty from the `sl` unit and deletes the specified SLIP interface from the list of network interfaces. For example, the following call will delete the first SLIP interface from the list of network interfaces:

```
slipDelete (0);
```

RETURNS OK, or ERROR if the unit number is invalid or uninitialized.

SEE ALSO `if_sl`

slipInit()

NAME *slipInit()* – initialize a SLIP interface

SYNOPSIS

```
STATUS slipInit
(
    int    unit,           /* SLIP device unit number (0 - 19) */
    char * devName,       /* name of the tty device to be initialized */
    char * myAddr,        /* address of the SLIP interface */
    char * peerAddr,      /* address of the remote peer SLIP interface */
    int    baud,          /* baud rate of SLIP device: 0=don't set rate */
    BOOL   compressEnable, /* explicitly enable CSLIP compression */
    BOOL   compressAllow, /* enable CSLIP compression on Rx */
    int    mtu            /* user set-able MTU */
)
```

DESCRIPTION This routine initializes a SLIP device. Its parameters specify the name of the tty device, the Internet addresses of both sides of the SLIP point-to-point link (i.e., the local and remote sides of the serial line connection), and CSLIP options.

The Internet address of the local side of the connection is specified in *myAddr* and the name of its tty device is specified in *devName*. The Internet address of the remote side is

smIfAttach()

specified in *peerAddr*. If *baud* is not zero, the baud rate will be the specified value; otherwise, the default baud rate will be the rate set by the tty driver. The *unit* parameter specifies the SLIP device unit number. Up to twenty units may be created.

The CLSIP options parameters *compressEnable* and *compressAllow* determine support for TCP/IP header compression. If *compressAllow* is TRUE (1), then CSLIP will be enabled only if a CSLIP type packet is received by this device. If *compressEnable* is TRUE (1), then CSLIP compression will be enabled explicitly for all transmitted packets, and compressed packets can be received.

The MTU option parameter allows the setting of the MTU for the link.

For example, the following call initializes a SLIP device, using the console's second port, where the Internet address of the local host is 192.10.1.1 and the address of the remote host is 192.10.1.2. The baud rate will be the default rate for /tyCo/1. CLSIP is enabled if a CSLIP type packet is received. The MTU of the link is 1006.

```
slipInit (0, "/tyCo/1", "192.10.1.1", "192.10.1.2", 0, 0, 1, 1006);
```

RETURNS OK, or ERROR if the device cannot be opened, memory is insufficient, or the route is invalid.

SEE ALSO [if_sl](#)

smIfAttach()

NAME *smIfAttach()* – publish the **sm** interface and initialize the driver and device

SYNOPSIS STATUS *smIfAttach*

```
(
int      unit,          /* interface unit number */
SM_ANCHOR * pAnchor,   /* local addr of anchor */
int      maxInputPkts, /* max no. of input pkts */
int      intType,      /* method of notif. */
int      intArg1,      /* interrupt argument #1 */
int      intArg2,      /* interrupt argument #2 */
int      intArg3,      /* interrupt argument #3 */
int      ticksPerBeat, /* heartbeat freq. */
int      numLoan       /* no. of buffers to loan */
)
```

DESCRIPTION This routine attaches an **sm** Ethernet interface to the network, if the interface exists. This routine makes the interface available by filling in the network interface record. The system will initialize the interface when it is ready to accept packets.

The shared memory region must have been initialized, via *smPktSetup()*, prior to calling this routine (typically by an OS-specific initialization routine). The *smIfAttach()* routine can be called only once per unit number.

The *pAnchor* parameter is the local address by which the local CPU may access the shared memory anchor.

The *maxInputPkts* parameter specifies the maximum number of incoming shared memory packets which may be queued to this CPU at one time.

The *intType*, *intArg1*, *intArg2*, and *intArg3* parameters allow a CPU to announce the method by which it is to be notified of input packets which have been queued to it.

The *ticksPerBeat* parameter specifies the frequency of the shared memory anchor's heartbeat. The frequency is expressed in terms of the number of CPU ticks on the local CPU corresponding to one heartbeat period.

If *numLoan* is non-zero, it specifies the number of shared memory packets available to be loaned out.

RETURNS OK or ERROR.

SEE ALSO *if_sm*

smMemAddToPool()

NAME *smMemAddToPool()* – add memory to the shared memory system partition (VxMP Opt.)

SYNOPSIS

```
STATUS smMemAddToPool
(
    char *   pPool, /* pointer to memory pool */
    unsigned poolSize /* block size in bytes */
)
```

DESCRIPTION This routine adds memory to the shared memory system partition after the initial allocation of memory. The memory added need not be contiguous with memory previously assigned, but it must be in the same address space.

pPool is the global address of shared memory added to the partition. The memory area pointed to by *pPool* must be in the same address space as the shared memory anchor and shared memory pool.

poolSize is the size in bytes of shared memory added to the partition.

AVAILABILITY This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

RETURNS OK, or ERROR if access to the shared memory system partition fails.

ERRNO S_smObjLib_LOCK_TIMEOUT

SEE ALSO smMemLib

smMemCalloc()

NAME *smMemCalloc()* – allocate memory for an array from the shared memory system partition (VxMP Opt.)

SYNOPSIS

```
void * smMemCalloc
(
    int elemNum, /* number of elements */
    int elemSize /* size of elements */
)
```

DESCRIPTION This routine allocates a block of memory for an array that contains *elemNum* elements of size *elemSize* from the shared memory system partition. The return value is the local address of the allocated shared memory block.

AVAILABILITY This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

RETURNS A pointer to the block, or NULL if the memory cannot be allocated.

ERRNO S_memLib_NOT_ENOUGH_MEMORY
S_smObjLib_LOCK_TIMEOUT

SEE ALSO smMemLib

smMemFindMax()

NAME *smMemFindMax()* – find the largest free block in the shared memory system partition (VxMP Opt.)

SYNOPSIS

```
int smMemFindMax (void)
```

| | |
|---------------------|---|
| DESCRIPTION | This routine searches for the largest block in the shared memory system partition free list and returns its size. |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | The size (in bytes) of the largest available block, or ERROR if the attempt to access the partition fails. |
| ERRNO | S_smObjLib_LOCK_TIMEOUT |
| SEE ALSO | smMemLib |

smMemFree()

| | |
|---------------------|--|
| NAME | <i>smMemFree()</i> – free a shared memory system partition block of memory (VxMP Opt.) |
| SYNOPSIS | <pre>STATUS smMemFree (void * ptr /* pointer to block of memory to be freed */)</pre> |
| DESCRIPTION | This routine takes a block of memory previously allocated with <i>smMemMalloc()</i> or <i>smMemCalloc()</i> and returns it to the free shared memory system pool. It is an error to free a block of memory that was not previously allocated. |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | OK, or ERROR if the block is invalid. |
| ERRNO | S_memLib_BLOCK_ERROR S_smObjLib_LOCK_TIMEOUT |
| SEE ALSO | smMemLib, <i>smMemMalloc()</i> , <i>smMemCalloc()</i> |

smMemMalloc()

| | |
|---------------------|---|
| NAME | <i>smMemMalloc()</i> – allocate a block of memory from the shared memory system partition (VxMP Opt.) |
| SYNOPSIS | <pre>void * smMemMalloc (unsigned nBytes /* number of bytes to allocate */)</pre> |
| DESCRIPTION | This routine allocates a block of memory from the shared memory system partition whose size is equal to or greater than <i>nBytes</i> . The return value is the local address of the allocated shared memory block. |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | A pointer to the block, or NULL if the memory cannot be allocated. |
| ERRNO | S_memLib_NOT_ENOUGH_MEMORY S_smObjLib_LOCK_TIMEOUT |
| SEE ALSO | smMemLib |

smMemOptionsSet()

| | |
|--------------------|--|
| NAME | <i>smMemOptionsSet()</i> – set the debug options for the shared memory system partition (VxMP Opt.) |
| SYNOPSIS | <pre>STATUS smMemOptionsSet (unsigned options /* options for system partition */)</pre> |
| DESCRIPTION | This routine sets the debug options for the shared system memory partition. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed or reallocated. In both cases, the following options can be selected for actions to be taken when an error is detected: (1) return the error status, (2) log an error message and return the error status, or (3) log an error message and suspend |

the calling task. These options are discussed in detail in the library manual entry for **smMemLib**.

| | |
|---------------------|---|
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | OK or ERROR. |
| ERRNO | S_smObjLib_LOCK_TIMEOUT |
| SEE ALSO | smMemLib |

smMemRealloc()

| | |
|---------------------|---|
| NAME | <i>smMemRealloc()</i> – reallocate a block of memory from the shared memory system partition (VxMP Opt.) |
| SYNOPSIS | <pre>void * smMemRealloc (void * pBlock, /* block to be reallocated */ unsigned newSize /* new block size */)</pre> |
| DESCRIPTION | This routine changes the size of a specified block and returns a pointer to the new block of shared memory. The contents that fit inside the new size (or old size, if smaller) remain unchanged. The return value is the local address of the reallocated shared memory block. |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | A pointer to the new block of memory, or NULL if the reallocation cannot be completed. |
| ERRNO | S_memLib_NOT_ENOUGH_MEMORY S_memLib_BLOCK_ERROR S_smObjLib_LOCK_TIMEOUT |
| SEE ALSO | smMemLib |

smMemShow()

NAME *smMemShow()* – show the shared memory system partition blocks and statistics (VxMP Opt.)

SYNOPSIS

```
void smMemShow
(
    int type /* 0 = statistics, 1 = statistics & list */
)
```

DESCRIPTION This routine displays the total amount of free space in the shared memory system partition, including the number of blocks, the average block size, and the maximum block size. It also shows the number of blocks currently allocated, and the average allocated block size.

If *type* is 1, it displays a list of all the blocks in the free list of the shared memory system partition.

WARNING This routine locks access to the shared memory system partition while displaying the information. This can compromise the access time to the partition from other CPUs in the system. Generally, this routine is used for debugging purposes only.

EXAMPLE

```
-> smMemShow 1
FREE LIST:
  num  addr      size
  ---  -
   1  0x4ffef0    264
   2  0x4fef18   1700
SUMMARY:
  status      bytes  blocks  ave block max block
  -
  current
    free     1964     2      982     1700
    alloc    2356     1     2356      -
  cumulative
    alloc    2620     2     1310      -
value = 0 = 0x0
```

AVAILABILITY This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

RETURNS N/A

SEE ALSO *smMemShow*, *windsh*, *Tornado User's Guide: Shell*

smNameAdd()

| | | | | | | | | | | | | | | | | |
|---------------------|---|-------------------------------|---|-------------------------|------------|---|---------------------------|------------|---|----------------------|--------------|---|-------------------------|------------|---|-------------------------------|
| NAME | <i>smNameAdd()</i> – add a name to the shared memory name database (VxMP Opt.) | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>STATUS smNameAdd (char * name, /* name string to enter in database */ void * value, /* value associated with name */ int type /* type associated with name */)</pre> | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>This routine adds a name of specified object type and value to the shared memory objects name database.</p> <p>The <i>name</i> parameter is an arbitrary null-terminated string with a maximum of 20 characters, including EOS.</p> <p>By convention, <i>type</i> values of less than 0x1000 are reserved by VxWorks; all other values are user definable. The following types are predefined in smNameLib.h :</p> <table><tr><td>T_SM_SEM_B</td><td>0</td><td>shared binary semaphore</td></tr><tr><td>T_SM_SEM_C</td><td>1</td><td>shared counting semaphore</td></tr><tr><td>T_SM_MSG_Q</td><td>2</td><td>shared message queue</td></tr><tr><td>T_SM_PART_ID</td><td>3</td><td>shared memory Partition</td></tr><tr><td>T_SM_BLOCK</td><td>4</td><td>shared memory allocated block</td></tr></table> <p>A name can be entered only once in the database, but there can be more than one name associated with an object ID.</p> | T_SM_SEM_B | 0 | shared binary semaphore | T_SM_SEM_C | 1 | shared counting semaphore | T_SM_MSG_Q | 2 | shared message queue | T_SM_PART_ID | 3 | shared memory Partition | T_SM_BLOCK | 4 | shared memory allocated block |
| T_SM_SEM_B | 0 | shared binary semaphore | | | | | | | | | | | | | | |
| T_SM_SEM_C | 1 | shared counting semaphore | | | | | | | | | | | | | | |
| T_SM_MSG_Q | 2 | shared message queue | | | | | | | | | | | | | | |
| T_SM_PART_ID | 3 | shared memory Partition | | | | | | | | | | | | | | |
| T_SM_BLOCK | 4 | shared memory allocated block | | | | | | | | | | | | | | |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. | | | | | | | | | | | | | | | |
| RETURNS | OK, or ERROR if there is insufficient memory for <i>name</i> to be allocated, if <i>name</i> is already in the database, or if the database is already full. | | | | | | | | | | | | | | | |
| ERRNO | S_smNameLib_NOT_INITIALIZED S_smNameLib_NAME_TOO_LONG S_smNameLib_NAME_ALREADY_EXIST S_smNameLib_DATABASE_FULL S_smObjLib_LOCK_TIMEOUT | | | | | | | | | | | | | | | |
| SEE ALSO | smNameLib , smNameShow | | | | | | | | | | | | | | | |

smNameFind()

| | |
|---------------------|---|
| NAME | <i>smNameFind()</i> – look up a shared memory object by name (VxMP Opt.) |
| SYNOPSIS | <pre>STATUS smNameFind (char * name, /* name to search for */ void * * pValue, /* pointer where to return value */ int * pType, /* pointer where to return object type */ int waitType /* NO_WAIT or WAIT_FOREVER */)</pre> |
| DESCRIPTION | <p>This routine searches the shared memory objects name database for an object matching a specified <i>name</i>. If the object is found, its value and type are copied to the addresses pointed to by <i>pValue</i> and <i>pType</i>. The value of <i>waitType</i> can be one of the following:</p> <p>NO_WAIT (0) The call returns immediately, even if <i>name</i> is not in the database.</p> <p>WAIT_FOREVER (-1) The call returns only when <i>name</i> is available in the database. If <i>name</i> is not already in, the database is scanned periodically as the routine waits for <i>name</i> to be entered.</p> |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | OK, or ERROR if the object is not found, if <i>name</i> is too long, or the wait type is invalid. |
| ERRNO | S_smNameLib_NOT_INITIALIZED S_smNameLib_NAME_TOO_LONG S_smNameLib_NAME_NOT_FOUND S_smNameLib_INVALID_WAIT_TYPE S_smObjLib_LOCK_TIMEOUT |
| SEE ALSO | smNameLib , smNameShow |

smNameFindByValue()

| | |
|---------------------|---|
| NAME | <i>smNameFindByValue()</i> – look up a shared memory object by value (VxMP Opt.) |
| SYNOPSIS | <pre>STATUS smNameFindByValue (void * value, /* value to search for */ char * name, /* pointer where to return name */ int * pType, /* pointer where to return object type */ int waitType /* NO_WAIT or WAIT_FOREVER */)</pre> |
| DESCRIPTION | <p>This routine searches the shared memory name database for an object matching a specified value. If the object is found, its name and type are copied to the addresses pointed to by <i>name</i> and <i>pType</i>. The value of <i>waitType</i> can be one of the following:</p> <p>NO_WAIT (0) The call returns immediately, even if the object value is not in the database.</p> <p>WAIT_FOREVER (-1) The call returns only when the object value is available in the database.</p> |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | OK, or ERROR if <i>value</i> is not found or if the wait type is invalid. |
| ERRNO | <pre>S_smNameLib_NOT_INITIALIZED S_smNameLib_VALUE_NOT_FOUND S_smNameLib_INVALID_WAIT_TYPE S_smObjLib_LOCK_TIMEOUT</pre> |
| SEE ALSO | <i>smNameLib</i> , <i>smNameShow</i> |

smNameRemove()

| | |
|---------------------|---|
| NAME | <i>smNameRemove()</i> – remove an object from the shared memory objects name database (VxMP Opt.) |
| SYNOPSIS | <pre>STATUS smNameRemove (char * name /* name of object to remove */)</pre> |
| DESCRIPTION | This routine removes an object called <i>name</i> from the shared memory objects name database. |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | OK, or ERROR if the object name is not in the database or if <i>name</i> is too long. |
| ERRNO | S_smNameLib_NOT_INITIALIZED S_smNameLib_NAME_TOO_LONG S_smNameLib_NAME_NOT_FOUND S_smObjLib_LOCK_TIMEOUT |
| SEE ALSO | smNameLib, smNameShow |

smNameShow()

| | |
|--------------------|---|
| NAME | <i>smNameShow()</i> – show the contents of the shared memory objects name database (VxMP Opt.) |
| SYNOPSIS | <pre>STATUS smNameShow (int level /* information level */)</pre> |
| DESCRIPTION | This routine displays the names, values, and types of objects stored in the shared memory objects name database. Predefined types are shown, using their ASCII representations; all other types are printed in hexadecimal. |

The *level* parameter defines the level of database information displayed. If *level* is 0, only statistics on the database contents are displayed. If *level* is greater than 0, then both statistics and database contents are displayed.

WARNING This routine locks access to the shared memory objects name database while displaying its contents. This can compromise the access time to the name database from other CPUs in the system. Generally, this routine is used for debugging purposes only.

EXAMPLE

```
-> smNameShow
Names in Database Max : 30 Current : 6 Free : 24
-> smNameShow 1
Names in Database Max : 30 Current : 6 Free : 24
Name                Value                Type
-----
inputImage           0x802340             SM_MEM_BLOCK
ouputImage           0x806340             SM_MEM_BLOCK
imagePool            0x802001             SM_MEM_PART
imageInSem           0x8e0001             SM_SEM_B
imageOutSem          0x8e0101             SM_SEM_C
actionQ              0x8e0201             SM_MSG_Q
userObject           0x8e0400             0x1b0
```

AVAILABILITY This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

RETURNS OK, or ERROR if the name facility is not initialized.

ERRNO S_smNameLib_NOT_INITIALIZED
S_smObjLib_LOCK_TIMEOUT

SEE ALSO smNameShow, smNameLib

smNetAttach()

NAME *smNetAttach()* – attach the shared memory network interface

SYNOPSIS

```
STATUS smNetAttach
(
    int          unit,          /* interface unit number */
    SM_ANCHOR * pAnchor,       /* addr of anchor */
    int          maxInputPkts, /* max queued packets */
    int          intType,       /* interrupt method */
```

smNetInetGet()

```

int          intArg1,      /* interrupt argument #1 */
int          intArg2,      /* interrupt argument #2 */
int          intArg3      /* interrupt argument #3 */
)

```

DESCRIPTION This routine attaches the shared memory interface to the network. It is called once by each CPU on the shared memory network. The *unit* parameter specifies the backplane unit number.

The *pAnchor* parameter is the local address by which the local CPU may access the shared memory anchor.

The *maxInputPkts* parameter specifies the maximum number of incoming shared memory packets which may be queued to this CPU at one time.

The *intType*, *intArg1*, *intArg2*, and *intArg3* parameters allow a CPU to announce the method by which it is to be notified of input packets which have been queued to it.

RETURNS OK, or ERROR if the shared memory interface cannot be attached.

SEE ALSO **smNetLib**

smNetInetGet()

NAME *smNetInetGet()* – get an address associated with a shared memory network interface

SYNOPSIS

```

STATUS smNetInetGet
(
    char * smName, /* device name */
    char * smInet, /* return inet */
    int   cpuNum  /* cpu number */
)

```

DESCRIPTION This routine returns the Internet address in *smInet* for the CPU specified by *cpuNum* on the shared memory network specified by *smName*. If *cpuNum* is NONE (-1), this routine returns information about the local (calling) CPU.

This routine can only be called after a call to *smNetAttach()*. It will block if the shared memory region has not yet been initialized.

This routine is only applicable if sequential addressing is being used over the backplane.

RETURNS OK, or ERROR if the Internet address cannot be found.

SEE ALSO **smNetLib**

smNetInit()

NAME `smNetInit()` – initialize the shared memory network driver

SYNOPSIS

```
STATUS smNetInit
(
    SM_ANCHOR * pAnchor,      /* local addr of anchor */
    char *      pMem,         /* local addr of shared memory */
    int         memSize,      /* size of shared memory */
    BOOL        tasType,      /* TRUE = hardware supports TAS */
    int         cpuMax,       /* max numbers of cpus */
    int         maxPktBytes,  /* size of data packets */
    u_long      startAddr    /* beginning address */
)
```

DESCRIPTION This routine is called once by the backplane master. It sets up and initializes the shared memory region of the shared memory network and starts the shared memory heartbeat. The *pAnchor* parameter is the local memory address by which the master CPU accesses the shared memory anchor. *pMem* contains either the local address of shared memory or the value NONE (-1), which implies that shared memory is to be allocated dynamically. *memSize* is the size, in bytes, of the shared memory region.

The *tasType* parameter specifies the test-and-set operation to be used to obtain exclusive access to the shared data structures. It is preferable to use a genuine test-and-set instruction, if the hardware permits it. In this case, *tasType* should be **SM_TAS_HARD**. If any of the CPUs on the backplane network do not support the test-and-set instruction, *tasType* should be **SM_TAS_SOFT**.

The *maxCpus* parameter specifies the maximum number of CPUs that may use the shared memory region.

The *maxPktBytes* parameter specifies the size, in bytes, of the data buffer in shared memory packets. This is the largest amount of data that may be sent in a single packet. If this value is not an exact multiple of 4 bytes, it will be rounded up to the next multiple of 4.

The *startAddr* parameter is only applicable if sequential addressing is desired. If *startAddr* is non-zero, it specifies the starting address to use for sequential addressing on the backplane. If *startAddr* is zero, sequential addressing is disabled.

RETURNS OK, or ERROR if the shared memory network cannot be initialized.

SEE ALSO `smNetLib`

smNetShow()

NAME *smNetShow()* – show information about a shared memory network

SYNOPSIS

```
STATUS smNetShow
(
    char * ifName, /* backplane interface name (NULL == "sm0") */
    BOOL  zero     /* TRUE = zap totals */
)
```

DESCRIPTION This routine displays information about the different CPUs configured in a shared memory network specified by *ifName*. It prints error statistics and zeros these fields if *zero* is set to TRUE.

EXAMPLE

```
-> smNetShow
Anchor at 0x800000
heartbeat = 705, header at 0x800010, free pkts = 237.
cpu int type   arg1      arg2      arg3      queued pkts
-----
0  poll                0x0      0x0      0x0      0
1  poll                0x0      0x0      0x0      0
2  bus-int             0x3      0xc9     0x0      0
3  mbox-2              0x2d     0x8000   0x0      0
input packets = 192    output packets = 164
output errors = 0     collisions = 0
value = 1 = 0x1
```

RETURNS OK, or ERROR if there is a hardware setup problem or the routine cannot be initialized.

SEE ALSO *smNetShow*

smObjAttach()

NAME *smObjAttach()* – attach the calling CPU to the shared memory objects facility (VxMP Opt.)

SYNOPSIS

```
STATUS smObjAttach
(
    SM_OBJ_DESC * pSmObjDesc /* pointer to shared memory descriptor */
)
```


| | |
|---------------------|---|
| DESCRIPTION | <p>This routine "attaches" the calling CPU to the shared memory objects facility. The shared memory area is identified by the shared memory descriptor with an address specified by <i>pSmObjDesc</i>. The descriptor must already have been initialized by calling <i>smObjInit()</i>.</p> <p>This routine is called automatically when the configuration macro <code>INCLUDE_SM_OBJ</code> is defined.</p> <p>This routine will complete the attach process only if and when the shared memory has been initialized by the master CPU. If the shared memory is not recognized as active within the timeout period (10 minutes), this routine returns <code>ERROR</code>.</p> <p>The <i>smObjAttach()</i> routine connects the shared memory objects handler to the shared memory interrupt. Note that this interrupt may be shared between the shared memory network driver and the shared memory objects facility when both are used at the same time.</p> |
| WARNING | <p>Once a CPU has attached itself to the shared memory objects facility, it cannot be detached. Since the shared memory network driver and the shared memory objects facility use the same low-level attaching mechanism, a CPU cannot be detached from a shared memory network driver if the CPU also uses shared memory objects.</p> |
| AVAILABILITY | <p>This routine is distributed as a component of the unbundled shared memory objects support option, <code>VxMP</code>.</p> |
| RETURNS | <p>OK, or <code>ERROR</code> if the shared memory objects facility is not active or the number of CPUs exceeds the maximum.</p> |
| ERRNO | <p><code>S_smLib_INVALID_CPU_NUMBER</code></p> |
| SEE ALSO | <p><i>smObjLib</i>, <i>smObjSetup()</i>, <i>smObjInit()</i></p> |

smObjGlobalToLocal()

| | |
|--------------------|---|
| NAME | <p><i>smObjGlobalToLocal()</i> – convert a global address to a local address (<code>VxMP</code> Opt.)</p> |
| SYNOPSIS | <pre>void * smObjGlobalToLocal (void * globalAdrs /* global address to convert */)</pre> |
| DESCRIPTION | <p>This routine converts a global shared memory address <i>globalAdrs</i> to its corresponding local value. This routine does not verify that <i>globalAdrs</i> is really a valid global shared memory address.</p> |

smObjInit()

| | |
|---------------------|---|
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | The local shared memory address pointed to by <i>globalAdrs</i> . |
| SEE ALSO | smObjLib |

smObjInit()

NAME *smObjInit()* – initialize a shared memory objects descriptor (VxMP Opt.)

SYNOPSIS

```
void smObjInit
(
    SM_OBJ_DESC * pSmObjDesc, /* ptr to shared memory descriptor */
    SM_ANCHOR * anchorLocalAdrs, /* shared memory anchor local adrs */
    int ticksPerBeat, /* cpu ticks per heartbeat */
    int smObjMaxTries, /* max no. of tries to obtain spinLock */
    int intType, /* interrupt method */
    int intArg1, /* interrupt argument #1 */
    int intArg2, /* interrupt argument #2 */
    int intArg3, /* interrupt argument #3 */
)
```

DESCRIPTION This routine initializes a shared memory descriptor. The descriptor must already be allocated in the CPU's local memory. Once the descriptor has been initialized by this routine, the CPU may attach itself to the shared memory area by calling *smObjAttach()*.

This routine is called automatically when the configuration macro **INCLUDE_SM_OBJ** is defined.

Only the shared memory descriptor itself is modified by this routine. No structures in shared memory are affected.

Parameters:

pSmObjDesc

the address of the shared memory descriptor to be initialized; this structure must be allocated before *smObjInit()* is called.

anchorLocalAdrs

the memory address by which the local CPU may access the shared memory anchor. This address may vary among CPUs in the system because of address offsets (particularly if the anchor is located in one CPU's dual-ported memory).

cpuNum

the number to be used to identify this CPU during shared memory operations. CPUs are numbered starting with zero for the master CPU, up to 1 less than the maximum number of CPUs defined during the master CPU's *smObjSetup()* call. CPUs can attach in any order, regardless of their CPU number.

ticksPerBeat

specifies the frequency of the shared memory anchor's heartbeat. The frequency is expressed in terms of how many CPU ticks on the local CPU correspond to one heartbeat period.

smObjMaxTries

specifies the maximum number of tries to obtain access to an internal mutually exclusive data structure. Its default value is 100, but it can be set to a higher value for a heavily loaded system.

intType, intArg1, intArg2, and intArg3

allow a CPU to announce the method by which it is to be notified of shared memory events. See the manual entry for *if_sm* for a discussion about interrupt types and their associated parameters.

- AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.
- RETURNS** N/A
- SEE ALSO** *smObjLib, smObjSetup(), smObjAttach()*

smObjLibInit()

- NAME** *smObjLibInit()* – install the shared memory objects facility (VxMP Opt.)
- SYNOPSIS** `STATUS smObjLibInit (void)`
- DESCRIPTION** This routine installs the shared memory objects facility. It is called automatically when the configuration macro `INCLUDE_SM_OBJ` is defined.
- AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.
- RETURNS** OK, or ERROR if the shared memory objects facility has already been installed.
- SEE ALSO** *smObjLib*

smObjLocalToGlobal()

| | |
|---------------------|---|
| NAME | <i>smObjLocalToGlobal()</i> – convert a local address to a global address (VxMP Opt.) |
| SYNOPSIS | <pre>void * smObjLocalToGlobal (void * localAdrs /* local address to convert */)</pre> |
| DESCRIPTION | This routine converts a local shared memory address <i>localAdrs</i> to its corresponding global value. This routine does not verify that <i>localAdrs</i> is really a valid local shared memory address. |
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | The global shared memory address pointed to by <i>localAdrs</i> . |
| SEE ALSO | <i>smObjLib</i> |

smObjSetup()

| | |
|--------------------|---|
| NAME | <i>smObjSetup()</i> – initialize the shared memory objects facility (VxMP Opt.) |
| SYNOPSIS | <pre>STATUS smObjSetup (SM_OBJ_PARAMS * smObjParams /* setup parameters */)</pre> |
| DESCRIPTION | <p>This routine initializes the shared memory objects facility by filling the shared memory header. It must be called only once by the shared memory master CPU (processor number 0). It is called automatically only by the master CPU, when the configuration macro <code>INCLUDE_SM_OBJ</code> is defined.</p> <p>Any CPU on the system backplane can use the shared memory objects facility; however, the facility must first be initialized on the master CPU. Then before other CPUs are attached to the shared memory area by <i>smObjAttach()</i>, each must initialize its own shared memory objects descriptor using <i>smObjInit()</i>. This mechanism is similar to the one used by the shared memory network driver.</p> |

The *smObjParams* parameter is a pointer to a structure containing the values used to describe the shared memory objects setup. This structure is defined as follows in *smObjLib.h*:

```
typedef struct sm_obj_params /* setup parameters */
{
    BOOL        allocatedPool; /* TRUE if shared memory pool is malloced */
    SM_ANCHOR * pAnchor;      /* shared memory anchor */
    char *      smObjFreeAdrs; /* start address of shared memory pool */
    int         smObjMemSize;  /* memory size reserved for shared memory */
    int         maxCpus;      /* max number of CPUs in the system */
    int         maxTasks;     /* max number of tasks using smObj */
    int         maxSems;      /* max number of shared semaphores */
    int         maxMsgQueues; /* max number of shared message queues */
    int         maxMemParts;  /* max number of shared memory partitions */
    int         maxNames;     /* max number of names of shared objects */
} SM_OBJ_PARAMS;
```

| | |
|---------------------|---|
| AVAILABILITY | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| RETURNS | OK, or ERROR if the shared memory pool cannot hold all the requested objects or the number of CPUs exceeds the maximum. |
| ERRNO | S_smObjLib_TOO_MANY_CPU S_smObjLib_SHARED_MEM_TOO_SMALL |
| SEE ALSO | smObjLib, <i>smObjInit()</i> , <i>smObjAttach()</i> |

smObjShow()

| | |
|--------------------|--|
| NAME | <i>smObjShow()</i> – display the current status of shared memory objects (VxMP Opt.) |
| SYNOPSIS | STATUS <i>smObjShow</i> () |
| DESCRIPTION | This routine displays useful information about the current status of shared memory objects facilities. |
| WARNING | The information returned by this routine is not static and may be obsolete by the time it is examined. This information is generally used for debugging purposes only. |
| EXAMPLE | <pre>-> smObjShow Shared Mem Anchor Local Addr: 0x600.</pre> |

```
Shared Mem Hdr Local Addr:  0xb1514.  
Attached CPU :              5  
Max Tries to Take Lock:    1  
Shared Object Type      Current  Maximum  Available  
-----  
Tasks                   1      20      19  
Binary Semaphores       8      30      20  
Counting Semaphores     2      30      20  
Messages Queues         3      10      7  
Memory Partitions       1       4      3  
Names in Database      16     100     84
```

- AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.
- RETURNS** OK, or ERROR if no shared memory objects are initialized.
- ERRNO** **S_smObjLib_NOT_INITIALIZED**
S_smLib_NOT_ATTACHED
- SEE ALSO** **smObjShow, smObjLib**

smObjTimeoutLogEnable()

- NAME** *smObjTimeoutLogEnable()* – enable/disable logging of failed attempts to take a spin-lock (VxMP Opt.)
- SYNOPSIS**
- ```
void smObjTimeoutLogEnable
(
 BOOL timeoutLogEnable /* TRUE to enable, FALSE to disable */
)
```
- DESCRIPTION** This routine enables or disables the printing of a message when an attempt to take a shared memory spin-lock fails.  
By default, message logging is enabled.
- AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.
- RETURNS** N/A
- SEE ALSO** **smObjLib**

---

## *sn83932EndLoad()*

- NAME** *sn83932EndLoad()* – initialize the driver and device
- SYNOPSIS**
- ```
END_OBJ * sn83932EndLoad
(
    char * initString /* String to be parse by the driver. */
)
```
- DESCRIPTION** This routine initializes the driver and the device to the operational state. All of the device specific parameters are passed in the *initString* parameter. This string must be of the format:
- unit_number:device_reg_addr:ivec*
- These parameters are all individually described in the *sn83932End* man page.
- RETURNS** An END object pointer or NULL on error.
- SEE ALSO** *sn83932End*

snattach()

- NAME** *snattach()* – publish the **sn** network interface and initialize the driver and device
- SYNOPSIS**
- ```
STATUS snattach
(
 int unit, /* unit number */
 char * pDevRegs, /* addr of device's regs */
 int ivec /* vector number */
)
```
- DESCRIPTION** This routine publishes the **sn** interface by filling in a network interface record and adding this record to the system list. It also initializes the driver and the device to the operational state.
- RETURNS** OK or ERROR.
- SEE ALSO** *if\_sn*

---

## **snmpMasterCleanup()**

**NAME** *snmpMasterCleanup()* – free up resources after a query times out

**SYNOPSIS**

```
void snmpMasterCleanup
(
 UINT_16_T reqid, /* request Id to track state block */
 UINT_16_T options /* as mentioned above */
)
```

**DESCRIPTION** If you use *snmpMasterHandlerAsync()*, the master agent calls this routine if the IPC layer determines that a timeout period for a query response has been exceeded. The *reqid* parameter is the same as the *requestId* value passed to the send routine. It is used to track the correct state block. The *options* parameter passes in a set of flags that control what actions the cleanup routine. Currently, there are three flags: SA\_CLEANUP\_INACTIVE, SA\_CLEANUP\_TIMEOUT, and SA\_CLEANUP\_CONTINUE. The continue and timeout flags should always be set. The inactive flag indicates that any objects associated with the subagent should be removed. Set this flag when the IPC layer determines that the subagent has stopped rather than timed out.

**RETURNS** N/A

**SEE ALSO** *subagentLib*

---

## **snmpMasterHandlerAsync()**

**NAME** *snmpMasterHandlerAsync()* – process messages from the subagent asynchronously

**SYNOPSIS**

```
void snmpMasterHandlerAsync
(
 OCTET_T * pMsg, /* pointer to the message */
 ALENGTH_T msgl, /* length of the message */
 IPCCOMP_T * pIpcComp, /* completion routine */
 IPCSEND_AS_T * pIpcSend, /* send routine */
 IPCRCV_T * pIpcRcv, /* receive routine */
 IPCFREE_T * pIpcFree, /* free routine */
 IPCAYT_T * pIpcAyt, /* status check routine */
 PTR_T ipchandle, /* ipchandle for the IPC scheme used */
 PTR_T user_priv /* MIB tree identifier */
)
```



**DESCRIPTION**

This function provides support for an asynchronous communication scheme between the master agent and its subagents. The shipped version of WindNet SNMP does not call this function. Instead, it calls *snmpMasterHandlerWR()*, a function that supports a synchronous communication scheme. If you want master agents and subagents to use an asynchronous communication scheme, you must rewrite *snmpQueMonitor()* to call *snmpMasterHandlerAsync()* instead of *snmpMasterHandlerWR()*. In addition, because *snmpMasterHandlerAsync()* does not return a function value, you will need to remove the *snmpQueMonitor()* code that responded to the *snmpMasterHandlerWR()* function value. The functionality handled by the removed code should instead be implemented in the function referenced by the *ipcComp* parameter. Use the parameters as follows:

*pMsg*

Expects a pointer to an **EBUFFER\_T** structure containing the data part of the message from the subagent. The message shows up on the queue as an **SA\_MESSAGE\_T** structure. The message expected by this parameter is contained in the **mesg** member of this structure. To extract this pointer, use **EbufferStart** macro defined in **buffer.h**.

*msgl*

Expects the length of the message referenced in *pMsg*. To retrieve this length value, use the **EBufferUsed** macro defined in **buffer.h**.

*pIpcComp*

Expects a pointer to the completion function, which must be of the form:

```
void masterIpcComp
(
 OCTET_T opcode, /* this specifies what needs to be done */
 EBUFFER_T * ebuf, /* reply message to be sent */
 VBL_T * vblst, /* list of varbinds that the message contained */
 PTR_T ipchandle /* subagent address */
)
```

The master agent executes this function upon completing processing for an unsolicited control message from a subagent (primarily registration requests, although a trap from the subagent will eventually find its way to this function). Your *masterIpcComp()* should be able handle things such as letting the subagent know the completion status of message it sent to the master agent.

For a registration routine, it must send the message in *ebuf* back to the subagent. This message contains the group ID of the MIB variables added to the master agent's MIB tree. The subagent needs this ID to make a deregistration request.

If you decide to support traps from subagents, this function must be able to forward the varbind list in *vblst* to the SNMP manager. In addition, it is your responsibility to acquire any values not specified in *vblst* and include it in the message you send the to the SNMP manager. Use the *opcode* to know when you are handling the completion processing for a registration request, a deregistration request, or a trap from a subagent.

For an example of an IPC completion routine, see *masterIpcComp()* defined in **masterIoLib.c**.

*pIpcSend*

Expects a pointer to the function that method routines should use to send messages to the subagent. This function must be of the form:

```
INT_32_T masterIpcSend
(
 EBUFFER_T * pBuf, /* message to be sent */
 PTR_T ipchandle /* address of subagent */
 UINT_16_T reqid /* ID for request sent */
)
```

To make the communication between the master agent and subagent asynchronous, this send routine should send the message to the subagent and return. Eventually, a response shows up on the master agent's local queue, or the query times out. How you process a query response or a query time out is almost entirely up to you.

To process a query response, you must call *snmpMasterQueryHandler()*. This function will handle the details of integrating the message from the subagent into a message to the SNMP manager.

To clean up after a send that times out, you must call *snmpMasterCleanup()*. The specifics of the mechanism you use are up to you, but you will likely need to integrate the mechanism with your *masterIpcSend()* routine. That is because this function gets the request ID that you will need for clean up. The request ID is a number generated internally to the SNMP master agent. It passes this value into your *masterIpcSend()* using the *reqid* parameter. To clean up after a send that times out, you submit the *reqid* in a call to *snmpMasterCleanup()*.

For an example of an *masterIpcSend()*, see the *masterIpcSend()* defined in **masterIoLib.c**.

*pIpcRcv*

This parameter is not used by *snmpMasterHandlerAsync()* and so should be null. It is included to maintain parallelism with *snmpMasterHandlerWR()*.

*pIpcFree*

Expects a pointer to a function of the form:

```
void masterIpcFree (PTR_T ipchandle)
```

The master agent uses this function to free any resources it might have allocated to maintain the IPC link with the subagent. The master agent calls this function when a subagent deregisters.

*pIpcAyt*

Expects a pointer to the function the master agent can use to test the connection with the subagent. This function must be of the form:

```
INT_32_T masterIpcAyt (PTR_T ipchandle)
```

For an example of such a function, see the *masterIpcAyt()* defined in **masterIoLib.c**.

*ipchandle*

Expects a pointer to the IPC handle used to access the subagent that sent this message. In the shipped implementation, this is a pointer to a message queue.

*user\_priv*

Expects a pointer to the MIB tree from which registration and deregistration requests want to add or delete objects or instances. If this pointer is NULL, the default MIB tree specified by *mib\_root\_node* is used.

**RETURNS** N/A

**SEE ALSO** **subagentLib**

## ***snmpMasterHandlerWR()***

**NAME** *snmpMasterHandlerWR()* – synchronous version of *snmpMasterHandlerAsync()*

**SYNOPSIS**

```
INT_32_T snmpMasterHandlerWR
(
 OCTET_T * pMsg, /* pointer to the message */
 ALENGTH_T msgl, /* length of the message */
 IPCSEND_T * pIpcSend, /* send routine */
 IPCRCV_T * pIpcRcv, /* receive routine */
 IPCFREE_T * pIpcFree, /* free routine */
 IPCAYT_T * pIpcAyt, /* status Check Routine */
 PTR_T ipchandle, /* ipchandle for the IPC scheme used */
 EBUFFER_T * pBuf, /* buffer to place reply in */
 VBL_T * pVblist, /* place to put varbinds */
 PTR_T user_priv /* MIB tree identifier */
)
```

**DESCRIPTION**

This function is called to process the control messages received from subagents when the communication method between master and subagent is synchronous.

To process a registration request, this function extracts the objects from the message and adds them as a group to the master agent's MIB tree. The actual get, test, and set methods for these objects reside in the subagent. To set up local methods for these routines, *snmpMasterHandlerAsync()* uses the function referenced in *pIpcSend* and *pIpcRcv*.

The methods local to the master agent use *pIpcSend* to send queries to the subagent which locally executes the actual method routine for the object. The subagent then transmits the results back to the master agent's public queue. When the function monitoring this queue

sees the query response, it transfers the message to the master agent's local queue where the *pIpcRcv* function is waiting for the response.

To process a deregistration request, this function extracts a group ID from the message and removes that group of objects from the master agent's MIB tree. It also executes the function in *pIpcFree* to free any resources allocated locally to maintain the IPC link with the deregistered subagent.

The *snmpMasterHandlerWR()* routine returns information using the output parameters *pBuf* and *pVblist* and its function return value. If the returned function value indicates success, the master agent sends the message returned in *pBuf* to the subagent that sent the registration or deregistration request. If the returned value of this function indicates failure, the master agent silently drops the packet.

This function as has the ability to return an opcode value, although this functionality is unused in the shipped version of WindNet SNMP. In fact, if *snmpMasterHandlerWR()* were to return an opcode, the current implementation of the master agent would silently drop the packet. The possibility of returning an opcode is supported to make it possible for you to create subagents that send traps. In this case, *snmpMasterHandlerWR()* would return an opcode and a varbind list using the *pVblist* parameter. You could then rewrite *snmpQueMonitor()*, the master agent function that calls *snmpMasterHandlerWR()*, so that it responds appropriately to the returned opcode and forwards the contents of *pVblist* to the SNMP manager.

Use the *snmpMasterHandlerWR()* parameters as follows:

*pMsg*

Expects a pointer to an **EBUFFER\_T** structure containing the data part of the message from the subagent. The message shows up on the queue as an **SA\_MESSAGE\_T** structure. The message expected by this parameter is contained in the **msg** member of the **SA\_MESSAGE\_T** structure. To extract this pointer, you can use the **EbufferStart** macro defined in defined in **buffer.h**.

*msgl*

Expects the length of the message referenced in *pMsg*. To retrieve this length value, use the **EbufferUsed** macro defined in **buffer.h**.

*pIpcSend*

Expects a pointer to the function that method routines should use to send messages to the subagent. This function must be of the form:

```
INT_32_T masterIpcSend
(
 EBUFFER_T * pBuf, /* message to be sent */
 PTR_T ipchandle /* address of subagent */
)
```

If *snmpMasterHandlerWR()* is processing a registration request from the subagent, it associates this function pointer with the group of objects it adds to the master agent's MIB tree. The methods for those objects call this routine to send a message to the

subagent to make a test, get, or set query against those variables. After using this function to send the message, the master agent then calls the function referenced in *pIpcRcv*. The *pIpcRcv* function waits on a local queue for a response from the subagent. For an example of an *masterIpcSend()* routine, see the *masterIpcSend()* defined in **masterIoLib.c**.

#### *ipcRcv*

Expects a pointer to a function of the form:

```
INT_32_T masterIpcRcv
(
 EBUFFER_T * pBuf, /* buffer to receive message */
 PTR_T ipchandle /* address of subagent */
)
```

If *snmpMasterHandlerWR()* is processing a registration request from the subagent, it associates this function pointer with the group of objects it adds to the master agent's MIB tree. The methods for those objects call this routine to wait on a local queue for a response from the subagent. For an example of an *masterIpcRcv()*, see the *masterIpcRcv()* defined in **masterIoLib.c**.

#### *ipcFree*

Expects a pointer to a function of the form:

```
void masterIpcFree (PTR_T ipchandle)
```

The master agent uses this function to free any resources it allocated to maintain the IPC link with the subagent. The master agent calls this function when a subagent deregisters.

#### *pIpcAyt*

Expects a pointer to the function the master agent can use to test the connection with the subagent. This function must be of the form:

```
INT_32_T masterIpcAyt (PTR_T ipchandle)
```

For an example of such a function, see the *masterIpcAyt()* defined in **masterIoLib.c**.

#### *ipchandle*

Expects a pointer to the IPC handle used to access the subagent that sent this message. In the shipped implementation, this is a pointer to a message queue.

#### *pBuf*

Expects a pointer to a previously allocated **EBUFFER\_T**. This is an output parameter that *snmpMasterHandlerWR()* uses this to return a reply packet, if one is generated. For example, if *snmpMasterHandlerWR()* successfully processes a registration request, it writes a message to the **EBUFFER\_T** at *pBuf*. This message contains the group ID for the objects just added to the master agent's MIB tree. When control returns from *snmpMasterHandlerWR()*, you must transmit this message back to the subagent, which will store the group ID for use in a deregistration request. In the current implementation, *snmpQueMonitor()* already handles this for you.

*pVblist*

Expects a pointer to a previously allocated **VBL\_T**. The intended use of this parameter is to provide an output vehicle for the varbind list received in a trap message from a subagent. Because of the application-dependent nature of traps, the shipped implementation of **snmpQueMonitor()** just drops the packet. However, if you want to support traps from your subagents, you can modify **snmpQueMonitor()** to check the returned value of **snmpMasterHandlerWR()** to watch for a trap message. You can then use **snmpIoTrapSend()** to forward the trap message in *pVblist* to the SNMP manager.

*user\_priv*

Expects a pointer to the MIB tree from which registration and deregistration requests want to add or delete objects or instances. If this pointer is **NULL**, the default MIB tree specified by **mib\_root\_node** is used.

If the message is trap request, it is the responsibility of the user code to acquire any values not specified in the trap message and to send the trap to the manager.

**RETURNS**

The opcode from the decoded packet or 0 or -1. An returned value of 0 indicates an error for which you should just drop the packet. A return value of -1 indicates success.

If this function returns an opcode, a value from 1 to 127, the shipped implementation just drops the packet. However, to support traps from the subagent, you could modify **snmpQueMonitor()** to note a returned value of **SA\_TRAP\_REQUEST** and then forward the varbind list in *pVblist* to the SNMP manager.

**SEE ALSO**

**subagentLib**

---

## **snmpMasterQueryHandler()**

**NAME**

**snmpMasterQueryHandler()** – handles replies from the subagent

**SYNOPSIS**

```
UINT_16_T snmpMasterQueryHandler
(
 OCTET_T * pMsg, /* pointer to the packet */
 ALENGTH_T msgl, /* length of packet */
 int flag /* should be 1 */
)
```

**DESCRIPTION**

This routine is for use with **snmpMasterHandlerAsync()**. It handles the replies to queries generated by the method routines. It decodes the message and tries to integrate the response with an outstanding packet. The *pMsg* and *msgl* parameters are pointers to the message and the length respectively. The *flag* parameter specifies whether the continuation routines should be run. This should always be set to 1.

**RETURNS** The request ID if routine could decode the packet or 0 in case of error.

**SEE ALSO** `subagentLib`

2

---

## *snmpMonitorSpawn()*

**NAME** *snmpMonitorSpawn()* – spawn `tMonQue` to run *snmpQueueMonitor()*

**SYNOPSIS** `void snmpMonitorSpawn (void)`

**DESCRIPTION** This function spawns the `tMonQue` task to run *snmpQueueMonitor()* a function that waits on the message queue that subagents use to leave messages for the master agent. The *snmpQueueMonitor()* waits forever on the master agent's message queue. When message comes in, it is interpreted using an `SA_MESSAGE_T` structure, which is defined in `ipcLib.h` as:

```
typedef struct SA_MESSAGE_S
{
 int msgType;
 MSG_Q_ID saId;
 EBUFFER_T mesg;
} SA_MESSAGE_T;
```

A switch internal to *snmpQueueMonitor()* handles the message according to the value of the `msgType` member.

If the message type is `CALL_QUERY_HANDLER`, the message is a response to a query from the master agent. The buffer referenced in the `mesg` is then transferred to the local message queue monitored by `tSnmpd`, where a *masterIpcRcv()* routine is waiting for a query response from a subagent.

If the message type is `CALL_REG_HANDLER`, the message is a control message such as a registration request, a deregistration request, or a trap. To respond to such requests, *snmpQueueMonitor()* passes the buffer in `mesg` to *snmpMasterHandlerWR()*.

If the message in the buffer passed to *snmpMasterHandlerWR()* is not correctly formed, the returned function value indicates failure and *snmpQueueMonitor()* drops the packet.

If the buffer passed to *snmpMasterHandlerWR()* is a correctly formed registration request, *snmpMasterHandlerWR()* adds the specified objects to the master agent's MIB tree. If the buffer contains a correctly formed deregistration request, *snmpMasterHandlerWR()* removes the specified objects from the master agent's MIB tree. In both cases the returned value of *snmpMasterHandlerWR()* indicates success and its *rbuf* parameter contains a message that *snmpQueueMonitor()* forwards to the subagent that sent the message.

**snmpSaHandlerAsync()**

In the case of a successful registration request, the message sent to the subagent contains a group ID for the objects just added to the master agent's MIB tree. When the subagent deregisters itself, it includes this ID in its deregistration message to the master agent. It also uses this group ID when it must register instances of the object just registered.

If the buffer passed to *snmpMasterHandlerWR()* contains a trap, the returned function value is *SA\_TRAP\_REQUEST*, the value extracted from the *opcode2* member of the header associated with the message. The message itself (minus the header) is a varbind list. It is returned using the *vbl* parameter. The current implementation of *snmpQueMonitor()* just drops this message. However, you can rewrite *snmpQueMonitor()* to make a *snmpIoTrapSend()* that forwards the varbind list to the SNMP manager. Likewise, you can implement appropriate responses to other *opcode2* values. Currently, *subagent.h* defines symbolic constants for opcodes 1 through 12 (with opcode 11, *SA\_TRAP\_REQUEST*, reserved for trap requests). If necessary you are free to use the remaining opcodes for message types specific to your implementation.

If your transport needs require that you rewrite *masterIoLib* to use an IPC other than message queues, you might need to modify this function, which is called from *snmpIoMain()* just before a call to *snmpIoBody()*. For example, if you use sockets as your IPC between the SNMP master agent and its subagents, *tSnmpd* could monitor the socket connection with the SNMP manager as well as the socket connections with the SNMP subagents.

**ASYNCHRONOUS COMMUNICATION**

The shipped version of *snmpQueMonitor()* uses *snmpMasterHandlerWR()* and thus processes messages asynchronously. However, if necessary, you can rewrite *snmpQueMonitor()* to call *snmpMasterHandlerAsync()* instead. For more information on *snmpMasterHandlerAsync()*, see its reference entry.

**RETURNS** N/A

**SEE ALSO** *masterIoLib*

---

## *snmpSaHandlerAsync()*

**NAME** *snmpSaHandlerAsync()* – asynchronous message processing routine for the subagent

**SYNOPSIS**

```
void snmpSaHandlerAsync
(
 OCTET_T * pMsg, /* message from the master-agent */
 ALENGTH_T msglength, /* length of message in octets */
 PTR_T root, /* root of mib tree */
 SA_IO_COMPLETE_T * pIoComp, /* IO completion routine */
)
```



```

SA_ERR_COMPLETE_T * pErrComp, /* error completion routine */
SA_REG_COMPLETE_T * pRegComp, /* registration complete routine */
PTR_T cookie /* cookie */
)

```

**DESCRIPTION**

It decodes the message in *pMsg* and responds appropriately, which can include testing, getting, and setting variables. After the message is processed, *snmpSaHandlerAsync()* then calls whichever completion routine is appropriate.

*pMsg*

Expects pointer to an octet string containing the message from the master agent.

*msglength*

Expects the length of the message.

*root*

Expects a pointer to the root of the subagent's MIB tree. If *root* is NULL, the default **mib\_root\_node** is used.

*pIoComp*

Expects a pointer to the function *snmpSaHandlerAsync()* should call after it has processed the message from the master agent. This routine should be able to send a response to the master agent, if necessary. This function must handle the building, encoding, transmission of the response to the master agent. This function must be of the form:

```
void SA_IO_COMPLETE_T(PTR_T pkt, SA_HEADER_T *hdr_blk, PTR_T cookie)
```

When the subagent calls this routine, it uses the *pkt* parameter to pass in a pointer to the data to be sent to the master agent. It uses the *hdr\_blk* parameter to pass in a pointer to the header to be included with the packet. It uses the *cookie* parameter to pass in the *cookies* specified in the call to *snmpSaHandlerAsync()*. You can use this *cookie* to carry information specific to your environment and application.

*pErrComp*

Expects a pointer to the function *snmpSaHandlerAsync()* should call if it cannot generate an appropriate response to a message from the master agent. This function must be of the form:

```
void SA_ERR_COMPLETE_T(int error_code, PTR_T cookie)
```

The *error\_code* passes in one of the following error codes:

- SA\_GEN\_ERROR
- SA\_UNKNOWN\_VERSION
- SA\_UNKNOWN\_OPCODE1
- SA\_UNKNOWN\_OPCODE2
- SA\_UNKNOWN\_ENCODING
- SA\_DECODE\_FAILURE
- SA\_ENCODE\_FAILURE
- SA\_UNKNOWN\_NODE

**snmpSaHandlerCleanup()**

SA\_UNKNOWN\_TAG  
 SA\_UNKNOWN\_GRP  
 SA\_SHORT\_MSG  
 SA\_IPC\_ERROR  
 SA\_LOCK\_ERROR  
 SA\_NODE\_ERROR  
 SA\_MEMORY\_ERROR  
 SA\_UNSUPPORTED\_TYPE  
 SA\_NO\_SAVED\_PACKET

The *cookie* parameter passes in the *cookie* specified in the call to *snmpSaHandlerAsync()*. You can use this *cookie* to carry information specific to your environment and application.

*pRegComp*

Expects a pointer to the function *snmpSaHandlerAsync()* should call in response to a registration completion message from the master agent. If successful, this message should contain a group ID for the MIB variables that the registration request added to the master agent's MIB tree. The subagent needs this ID when it comes time to deregister and remove those variables from the master agent's MIB tree. This function must be of the form:

```
void SA_REG_COMPLETE_T (INT_32_T ecode, SA_HEADER_T *hdr_blk,
 VBL_T *vblp, PTR_T cookie)
```

This completion routine expects an error code in *ecode*, a header block in *hdr\_blk*, a list of nodes at *vblp*, and the *cookie* passed into the *snmpSaHandlerAsync()*.

*cookie*

Expects a pointer that you can use to pass data unchanged to the functions you specified in the *pIoComp*, *pErrComp*, and *pRegComp* functions.

**RETURNS** N/A

**SEE ALSO** [subagentLib](#)

---

## **snmpSaHandlerCleanup()**

**NAME** *snmpSaHandlerCleanup()* – cleanup routine for subagent

**SYNOPSIS**

```
void snmpSaHandlerCleanup
(
 PTR_T pPkt, /* pointer to the packet */
 SA_HEADER_T * pHdr /* header block */
)
```

|                    |                                                                                                                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine is called by the IO completion routine if it detects an error. It either frees or arranges to free any resources that might have been allocated for processing a query from the master agent. The information at <i>pPkt</i> and <i>pHdr</i> is passed unchanged into the completion routine. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                        |
| <b>SEE ALSO</b>    | <i>subagentLib</i>                                                                                                                                                                                                                                                                                         |

---

## *snmpSaHandlerContinue()*

**NAME** *snmpSaHandlerContinue()* – subagent continuation function

**SYNOPSIS**

```
void snmpSaHandlerContinue
(
 SNMP_PKT_T * pPkt /* pointer to the SNMP packet */
)
```

**DESCRIPTION** This routine is similar to *snmpdContinue()*. Method routines that do not complete their tasks before returning should arrange to have this routine called when the task is finished. This routine should not be called if you call *snmpSaHandlerWR()*. The *pPkt* parameter expects a pointer to the packet. If *SNMP\_CONTINUE\_REENTRANT* is installed, this routine will attempt to release the per-packet write lock.

**RETURNS** N/A

**SEE ALSO** *subagentLib*

---

## *snmpSaHandlerFinish()*

**NAME** *snmpSaHandlerFinish()* – encode packet for subagent IO completion

**SYNOPSIS**

```
INT_32_T snmpSaHandlerFinish
(
 PTR_T pkt, /* pointer to the packet */
 SA_HEADER_T * pHdr, /* header block */
 EBUFFER_T * pBuf /* buffer to place the result in */
)
```

- DESCRIPTION** This routine encodes the packet at *pkt* and the header block at *pHdr*. If *pBuf* is empty, this routine tries to allocate space. If it cannot or if the space provided is too small, an error is returned.
- RETURNS** 0 on success, or a non-zero value on failure.
- SEE ALSO** **subagentLib**

---

## **snmpSaHandlerWR()**

**NAME** *snmpSaHandlerWR()* – provide *snmpSaHandlerAsync()* functionality synchronously

**SYNOPSIS**

```
INT_32_T snmpSaHandlerWR
(
 OCTET_T * pMsg, /* message from the master-agent */
 ALENGTH_T msgl, /* length of message in octets */
 EBUFFER_T * pBuf, /* buffer to hold reply packet */
 SA_HEADER_T * pHdr, /* place for header structure */
 VBL_T * pVblist, /* place for vblist */
 PTR_T root /* root of mib tree */
)
```

**DESCRIPTION** This routine puts a synchronous shell around *snmpSaHandlerAsync()*. Like *snmpSaHandlerAsync()*, this function can decode a message from the master agent. If the message is a query against a variable in the subagent's MIB tree, *snmpSaHandlerWR()* processes the request and generates a response. However, *snmpSaHandlerWR()* does not handle the completion processing for the message that would have been handled by the *pIoComp*, *pErrComp*, and *pRegComp* routines specified as input to *snmpSaHandlerAsync()*.

Instead, it uses its returned function value to indicate that status of the message processing and uses *pBuf*, *pHdr*, and *pVblist* as output parameters if that status requires additional processing on your part. For example, if the message was a successfully processed query, the response data is included in *pVblist* and a header is included in *pHdr*, but that response is not yet encoded in a packet or transmitted back to the master agent. In *snmpSaHandlerAsync()*, all that would normally be handled in the *pIoComp* routine. Effectively, you must now call your *pIoComp* routine explicitly.

*pMsg*

Expects a pointer to the message, an octet string, from the master agent.

*msgl*

Expects the length of the message starting at *pMsg*.

*pBuf*

Expects a pointer to a previously allocated **EBUFFER\_T** into which this function can write a response, if any. In some cases (if **opcode1** is **SA\_QUERY\_REQUEST**), instead of indicating an error in the returned value of *snmpSaHandlerWR()*, the error is encoded into this message. This is done for errors more appropriately handled by the SNMP manager.

*pHdr*

Expects a pointer to a previously allocated **SA\_HEADER\_T** structure into which this function can write header block information, if necessary. If **hdr\_blk.sa\_error** is non-zero, other members might not contain valid data.

*pVblist*

Expects a pointer to a previously allocated **VBL\_T** structure into which this function can write the list of nodes found in the original message from the master agent.

*root*

Expects a pointer to the root of the subagent's MIB tree. If *root* is **NULL**, the default **mib\_root\_node** is used.

**RETURNS**

0 on success, or a positive value indicating an error. For return code values, see **subagent.h**. Using these values as a switch, you should call one of the functions you would have specified for *pIoComp*, *pErrComp*, or *pRegComp* in a call to *snmpSaHandlerAsync()*.

**SEE ALSO****subagentLib**


---

## **snmpSaInit()**

**NAME***snmpSaInit()* – initialize the subagent**SYNOPSIS**

```
PTR_T snmpSaInit
(
 PTR_T saId, /* ipchandle for socket/queue */
 PTR_T sa_root, /* pointer to mib root node */
 SA_REG_COMPLETE_T saRegComp /* registration complete routine */
)
```

**DESCRIPTION**

Call this routine to initialize an SNMP subagent. Internally, this routine creates an IPC mechanism for receiving messages from the master agent and then spawns a task to run *snmpSaMonitor()*, a function that monitors the IPC mechanism created by *snmpSaInit()*. As input, *snmpSaInit()* takes the parameters: *saId*, *sa\_root*, and *saRegComp*.

**snmpSubEncode( )***salD*

Expects a null. In most functions in this library, an *salD* parameter is a pointer to the IPC mechanism used to pass messages to the subagent. However, the IPC mechanism is first created internally to this function. Thus, this *salD* parameter is not actually used for input nor is it an output parameter. It is included for parallelism with other functions in this library.

*sa\_root*

This parameter provides a pointer to the MIB tree for this subagent.

*saRegComp*

Use this routine to pass in a pointer to the function that *snmpSaHandlerAsync( )* should execute in response to a registration status message from the master agent. If the registration was successful, the response contains a group ID for the MIB variables registered with the master agent. You will need this group ID when it comes time to deregister this SNMP subagent, or when you need to register instances of the object just registered.

Although this function sets up the IPC mechanism and spawns the task that is effectively the SNMP subagent, this routine does not actually register the subagent with the master agent. The details of how and when one does that are entirely dependent upon the nature of the system you are designing. Thus, no generic registration utility is provided. For more information on sending a registration request to the master agent, see the description of *hdrBlkBuild( )*.

**RETURNS** A pointer to the IPC mechanism created within this function, or NULL on failure.

**SEE ALSO** [saIoLib](#)

**snmpSubEncode( )**

**NAME** *snmpSubEncode( )* – encode a packet for transmission to master agent or subagent

**SYNOPSIS**

```

INT_32_T snmpSubEncode
(
 VBL_T * pVblist, /* varbindlist to be encoded */
 SA_HEADER_T * pHdr, /* header block structure */
 SA_DEMUX_T * pDemuxer, /* demuxer structure */
 EBUFFER_T * pBuf /* buffer to place result in */
)

```

**DESCRIPTION** This routine encodes a memory-resident varbind list. The result is a buffer containing a message ready for transmission. Most of the arguments are values to be encoded into the buffer.

*pVblist*

Expects a pointer to a **VBL\_T** structure containing the list of the varbinds to be encoded in the message. In a control message, the varbinds identify the nodes or instances to be added or removed from the master agents MIB tree. In a query message, the varbinds identify the variables to be gotten or set. In a trap message sent from a subagent to its master agent, the varbinds specify the objects to be sent in a trap message to the SNMP manager. A trap message from a subagent follows the SNMPv2 trap style. Thus, the first object in the list must always be **sysUpTime**. The second object must be a **snmpTrapOID.0** whose value is the administratively assigned name of the notification.

*pHdr*

Expects a pointer to a **SA\_HEADER\_T** structure containing all the items that go into the message header.

*pDemuxer*

Expects a pointer to an **SA\_DEMUX\_T** structure containing all the information the subagent might need to demux the packet. That is, to determine the time and space contexts for this request. In a v1 request, the string part of the demuxer is the community string and the object ID is unused. In a v2 request, the string is the local entity string from the context and the Object ID is the local time ID from the context.

*pBuf*

Expects a pointer to an **EBUFFER\_T** structure into which *snmpSubEncode()* can write the encoded packet. If *pBuf* references a previously allocated **EBUFFER\_T** structure, *snmpSubEncode()* uses that space. Otherwise, *snmpSubEncode()* tries to the necessary space.

**RETURNS** 0, if successful (that is, the structure at *pBuf* is ready for transmission); 1, if there is an illegal or unknown argument; 2, if there is insufficient buffer space at *pBuf* or space cannot be allocated.

**SEE ALSO** **subagentLib**

---

## ***sntpTimeGet()***

**NAME** *sntpTimeGet()* – retrieve the current time from a remote source

**SYNOPSIS**

```
STATUS sntpTimeGet
(
 char * pServerAddr, /* server IP address or hostname */
 u_int timeout, /* timeout interval in ticks */
 struct timespec * pCurrTime /* storage for retrieved time value */
)
```

**sntpClockSet()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine stores the current time as reported by an SNTP/NTP server in the location indicated by <i>pCurrTime</i> . The reported time is first converted to the elapsed time since January 1, 1970, 00:00, GMT, which is the base value used by UNIX systems. If <i>pServerAddr</i> is NULL, the routine listens for messages sent by an SNTP/NTP server in broadcast mode. Otherwise, this routine sends a request to the specified SNTP/NTP server and extracts the reported time from the reply. In either case, an error is returned if no message is received within the interval specified by <i>timeout</i> . Typically, SNTP/NTP servers operating in broadcast mode send update messages every 64 to 1024 seconds. An infinite timeout value is specified by <b>WAIT_FOREVER</b> . |
| <b>RETURNS</b>     | OK, or ERROR if unsuccessful.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>ERRNO</b>       | <b>S_sntpLib_INVALID_PARAMETER</b><br><b>S_sntpLib_INVALID_ADDRESS</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <b>sntpLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

## **sntpClockSet()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>sntpClockSet()</i> – assign a routine to access the reference clock                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre> STATUS sntpClockSet (     FUNCPTR pClockHookRtn /* new interface to reference clock */ ) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | <p>This routine installs a hook routine that is called to access the reference clock used by the SNTP server. This hook routine must use the following interface:</p> <pre> STATUS sntpClockHook (int request, void *pBuffer); </pre> <p>The hook routine should copy one of three settings used by the server to construct outgoing NTP messages into <i>pBuffer</i> according to the value of the <i>request</i> parameter. If the requested setting is available, the installed routine should return OK (or ERROR otherwise).</p> <p>This routine calls the given hook routine with the <i>request</i> parameter set to <b>SNTPS_ID</b> to get the 32-bit reference identifier in the format specified in RFC 1769. It also calls the hook routine with <i>request</i> set to <b>SNTPS_RESOLUTION</b> to retrieve a 32-bit value containing the clock resolution in nanoseconds. That value will be used to determine the 8-bit signed integer indicating the clock precision (according to the format specified in RFC 1769). Other library routines will set the <i>request</i> parameter to <b>SNTPS_TIME</b> to retrieve the current 64-bit NTP timestamp from <i>pBuffer</i> in host byte order. The routine</p> |



*sntpNsecToFraction()* will convert a value in nanoseconds to the format required for the NTP fractional part.

**RETURNS** OK or ERROR.

**ERRNO** N/A

**SEE ALSO** *sntpLib*

---

## *sntpConfigSet()*

**NAME** *sntpConfigSet()* – change SNTP server broadcast settings

**SYNOPSIS**

```
STATUS sntpConfigSet
(
 int setting, /* configuration option to change */
 void * pValue /* new value for parameter */
)
```

**DESCRIPTION** This routine alters the configuration of the SNTP server when operating in broadcast mode. A *setting* value of `SNTPS_DELAY` interprets the contents of *pValue* as the new 16-bit broadcast interval. When *setting* equals `SNTPS_ADDRESS`, *pValue* should provide the string representation of an IP broadcast or multicast address (for example, "224.0.1.1"). Any changed settings will take effect after the current broadcast interval is completed and the corresponding NTP message is sent.

**RETURNS** OK or ERROR.

**ERRNO** `S_sntpLib_INVALID_PARAMETER`

**SEE ALSO** *sntpLib*

---

## **sntpsNsecToFraction()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sntpsNsecToFraction()</b> – convert portions of a second to NTP format                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>ULONG sntpsNsecToFraction (     ULONG nsecs /* nanoseconds to convert to binary fraction */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | This routine is provided for convenience in fulfilling an <b>SNTPS_TIME</b> request to the clock hook. It converts a value in nanoseconds to the fractional part of the NTP timestamp format. The routine is not designed to convert non-normalized values greater than or equal to one second. Although the NTP time format provides a precision of about 200 pico-seconds, rounding errors in the conversion process decrease the accuracy as the input value increases. In the worst case, only the 24 most significant bits are valid, which reduces the precision to tenths of a micro-second. |
| <b>RETURNS</b>     | Value for NTP fractional part in host-byte order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>ERRNO</b>       | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>sntpsLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

## **so()**

|                    |                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>so()</b> – single-step, but step over a subroutine                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>STATUS so (     int task /* task to step; 0 = use default */ )</pre>                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | This routine single-steps a task that is stopped at a breakpoint. However, if the next instruction is a JSR or BSR, <b>so()</b> breaks at the instruction following the subroutine call instead. To execute, enter:<br><br>-> <b>so [task]</b><br><br>If <i>task</i> is omitted or zero, the last task referenced is assumed. |
| <b>SEE ALSO</b>    | <b>dbgLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                          |

---

## *socket()*

**NAME** *socket()* – open a socket

**SYNOPSIS**

```
int socket
(
 int domain, /* address family (for example, AF_INET) */
 int type, /* SOCK_STREAM, SOCK_DGRAM, or SOCK_RAW */
 int protocol /* socket protocol (usually 0) */
)
```

**DESCRIPTION** This routine opens a socket and returns a socket descriptor. The socket descriptor is passed to the other socket routines to identify the socket. The socket descriptor is a standard I/O system file descriptor (fd) and can be used with the *close()*, *read()*, *write()*, and *ioctl()* routines.

Available socket types include:

**SOCK\_STREAM**

Specifies a connection-based (stream) socket.

**SOCK\_DGRAM**

Specifies a datagram (UDP) socket.

**SOCK\_RAW**

Specifies a raw socket.

**RETURNS** A socket descriptor, or ERROR.

**SEE ALSO** *sockLib*

---

## *sp()*

**NAME** *sp()* – spawn a task with default parameters

**SYNOPSIS**

```
int sp
(
 FUNCPTR func, /* function to call */
 int arg1, /* first of nine args to pass to spawned task */
 int arg2,
 int arg3,
 int arg4,
 int arg5,
```

**sprintf()**

```

 int arg6,
 int arg7,
 int arg8,
 int arg9
)

```

**DESCRIPTION** This command spawns a specified function as a task with the following defaults:

priority:

100

stack size:

20,000 bytes

task ID:

highest not currently used

task options:

VX\_FP\_TASK – execute with floating-point coprocessor support.

task name:

A name of the form **tN** where N is an integer which increments as new tasks are spawned, e.g., **t1**, **t2**, **t3**, etc.

The task ID is displayed after the task is spawned.

This command is a short form of the underlying *taskSpawn()* routine, convenient for spawning tasks in which the default parameters are satisfactory. If the default parameters are unacceptable, *taskSpawn()* should be called directly.

**RETURNS** A task ID, or ERROR if the task cannot be spawned.

**SEE ALSO** **usrLib**, **taskLib**, *taskSpawn()*, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

**sprintf()**

**NAME** *sprintf()* – write a formatted string to a buffer (ANSI)

**SYNOPSIS**

```

int sprintf
(
 char * buffer, /* buffer to write to */
 const char * fmt /* format string */
)

```

|                    |                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine copies a formatted string to a specified buffer, which is null-terminated. Its function and syntax are otherwise identical to <i>printf()</i> . |
| <b>RETURNS</b>     | The number of characters copied to <i>buffer</i> , not including the NULL terminator.                                                                        |
| <b>SEE ALSO</b>    | <i>fioLib, printf()</i> , <i>American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: Input/Output (stdio.h)</i>     |

---

## *spy()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>spy()</i> – begin periodic task activity reports                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>void spy (     int freq,          /* reporting freq in sec, 0 = default of 5 */     int ticksPerSec /* interrupt clock freq, 0 = default of 100 */ )</pre>                                                                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>This routine collects task activity data and periodically runs <i>spyReport()</i>. Data is gathered <i>ticksPerSec</i> times per second, and a report is made every <i>freq</i> seconds. If <i>freq</i> is zero, it defaults to 5 seconds. If <i>ticksPerSec</i> is omitted or zero, it defaults to 100.</p> <p>This routine spawns <i>spyTask()</i> to do the actual reporting.</p> <p>It is not necessary to call <i>spyClkStart()</i> before running <i>spy()</i>.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <i>usrLib, spyLib, spyClkStart(), spyTask(), VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                                                                                                                                    |

---

## *spyClkStart()*

|                 |                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <i>spyClkStart()</i> – start collecting task activity data                                          |
| <b>SYNOPSIS</b> | <pre>STATUS spyClkStart (     int intsPerSec /* timer interrupt freq, 0 = default of 100 */ )</pre> |

## ***spyClkStop()***

- DESCRIPTION** This routine begins data collection by enabling the auxiliary clock interrupts at a frequency of *intsPerSec* interrupts per second. If *intsPerSec* is omitted or zero, the frequency will be 100. Data from previous collections is cleared.
- RETURNS** OK, or ERROR if the CPU has no auxiliary clock, or if task create and delete hooks cannot be installed.
- SEE ALSO** *usrLib*, *spyLib*, *sysAuxClkConnect()*, *VxWorks Programmer's Guide: Target Shell*

---

## ***spyClkStop()***

- NAME** *spyClkStop()* – stop collecting task activity data
- SYNOPSIS** `void spyClkStop (void)`
- DESCRIPTION** This routine disables the auxiliary clock interrupts. Data collected remains valid until the next *spyClkStart()* call.
- RETURNS** N/A
- SEE ALSO** *usrLib*, *spyLib*, *spyClkStart()*, *VxWorks Programmer's Guide: Target Shell*

---

## ***spyHelp()***

- NAME** *spyHelp()* – display task monitoring help menu
- SYNOPSIS** `void spyHelp (void)`
- DESCRIPTION** This routine displays a summary of *spyLib* utilities:
- |                                           |                                                                     |
|-------------------------------------------|---------------------------------------------------------------------|
| <code>spyHelp</code>                      | Print this list                                                     |
| <code>spyClkStart [ticksPerSec]</code>    | Start task activity monitor running at ticksPerSec ticks per second |
| <code>spyClkStop</code>                   | Stop collecting data                                                |
| <code>spyReport</code>                    | Prints display of task activity statistics                          |
| <code>spyStop</code>                      | Stop collecting data and reports                                    |
| <code>spy [freq[,ticksPerSec]]</code>     | Start spyClkStart and do a report every freq seconds                |
| <code>ticksPerSec defaults to 100.</code> | <code>freq defaults to 5 seconds.</code>                            |

**RETURNS** N/A

**SEE ALSO** **usrLib**, **spyLib**, *VxWorks Programmer's Guide: Target Shell*

---

## ***spyLibInit()***

**NAME** *spyLibInit()* – initialize task cpu utilization tool package

**SYNOPSIS** `void spyLibInit (void)`

**DESCRIPTION** This routine initializes the task cpu utilization tool package. If the configuration macro `INCLUDE_SPY` is defined, it is called by the root task, *usrRoot()*, in `usrConfig.c`.

**RETURNS** N/A

**SEE ALSO** **spyLib**, **usrLib**

---

## ***spyReport()***

**NAME** *spyReport()* – display task activity data

**SYNOPSIS** `void spyReport (void)`

**DESCRIPTION** This routine reports on data gathered at interrupt level for the amount of CPU time utilized by each task, the amount of time spent at interrupt level, the amount of time spent in the kernel, and the amount of idle time. Time is displayed in ticks and as a percentage, and the data is shown since both the last call to *spyClkStart()* and the last *spyReport()*. If no interrupts have occurred since the last *spyReport()*, nothing is displayed.

**RETURNS** N/A

**SEE ALSO** **usrLib**, **spyLib**, *spyClkStart()*, *VxWorks Programmer's Guide: Target Shell*

## **spyStop()**

|                    |                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>spyStop()</i> – stop spying and reporting                                                                             |
| <b>SYNOPSIS</b>    | <code>void spyStop (void)</code>                                                                                         |
| <b>DESCRIPTION</b> | This routine calls <i>spyClkStop()</i> . Any periodic reporting by <i>spyTask()</i> is terminated.                       |
| <b>RETURNS</b>     | N/A                                                                                                                      |
| <b>SEE ALSO</b>    | <i>usrLib</i> , <i>spyLib</i> , <i>spyClkStop()</i> , <i>spyTask()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> |

---

## **spyTask()**

|                    |                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>spyTask()</i> – run periodic task activity reports                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>void spyTask (     int freq /* reporting frequency, in seconds */ )</pre>                                                                                            |
| <b>DESCRIPTION</b> | This routine is spawned as a task by <i>spy()</i> to provide periodic task activity reports. It prints a report, delays for the specified number of seconds, and repeats. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <i>usrLib</i> , <i>spyLib</i> , <i>spy()</i> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                            |

---

## **sqrt()**

|                 |                                                                                   |
|-----------------|-----------------------------------------------------------------------------------|
| <b>NAME</b>     | <i>sqrt()</i> – compute a non-negative square root (ANSI)                         |
| <b>SYNOPSIS</b> | <pre>double sqrt (     double x /* value to compute the square root of */ )</pre> |



|                      |                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b>   | This routine computes the non-negative square root of $x$ in double precision. A domain error occurs if the argument is negative. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                     |
| <b>RETURNS</b>       | The double-precision square root of $x$ or 0 if $x$ is negative.                                                                  |
| <b>ERRNO</b>         | EDOM                                                                                                                              |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                                                         |

---

## ***sqrtf()***

|                      |                                                                                  |
|----------------------|----------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>sqrtf()</i> – compute a non-negative square root (ANSI)                       |
| <b>SYNOPSIS</b>      | <pre>float sqrtf (     float x /* value to compute the square root of */ )</pre> |
| <b>DESCRIPTION</b>   | This routine returns the non-negative square root of $x$ in single precision.    |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                    |
| <b>RETURNS</b>       | The single-precision square root of $x$ .                                        |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                  |

---

## ***squeeze()***

|                    |                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>squeeze()</i> – reclaim fragmented free space on an RT-11 volume                           |
| <b>SYNOPSIS</b>    | <pre>STATUS squeeze (     char * devName /* RT-11 device to squeeze, e.g., "/fd0/" */ )</pre> |
| <b>DESCRIPTION</b> | This command moves data around on an RT-11 volume so that any areas of free space are merged. |

## ***sr()***

- NOTE** No device files should be open when this procedure is called. The subsequent condition of such files would be unknown and writing to them could corrupt the entire disk.
- RETURNS** OK, or ERROR if the device cannot be opened or squeezed.
- SEE ALSO** *usrLib*, *VxWorks Programmer's Guide: Target Shell*

---

## ***sr()***

- NAME** *sr()* – return the contents of the status register (MC680x0)
- SYNOPSIS**
- ```
int sr
(
    int taskId /* task ID, 0 means default task */
)
```
- DESCRIPTION** This command extracts the contents of the status register from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.
- RETURNS** The contents of the status register.
- SEE ALSO** *dbgArchLib*, *VxWorks Programmer's Guide: Target Shell*

sramDevCreate()

- NAME** *sramDevCreate()* – create a PCMCIA memory disk device
- SYNOPSIS**
- ```
BLK_DEV *sramDevCreate
(
 int sock, /* socket no. */
 int bytesPerBlk, /* number of bytes per block */
 int blksPerTrack, /* number of blocks per track */
 int nBlocks, /* number of blocks on this device */
 int blkOffset /* no. of blks to skip at start of device */
)
```
- DESCRIPTION** This routine creates a PCMCIA memory disk device.

**RETURNS** A pointer to a block device structure (`BLK_DEV`), or `NULL` if memory cannot be allocated for the device structure.

**SEE ALSO** `sramDrv`, `ramDevCreate()`

---

## *sramDrv()*

**NAME** `sramDrv()` – install a PCMCIA SRAM memory driver

**SYNOPSIS**

```
STATUS sramDrv
(
 int sock /* socket no. */
)
```

**DESCRIPTION** This routine initializes a PCMCIA SRAM memory driver. It must be called once, before any other routines in the driver.

**RETURNS** `OK`, or `ERROR` if the I/O system cannot install the driver.

**SEE ALSO** `sramDrv`

---

## *sramMap()*

**NAME** `sramMap()` – map PCMCIA memory onto a specified ISA address space

**SYNOPSIS**

```
STATUS sramMap
(
 int sock, /* socket no. */
 int type, /* 0: common 1: attribute */
 int start, /* ISA start address */
 int stop, /* ISA stop address */
 int offset, /* card offset address */
 int extraws /* extra wait state */
)
```

**DESCRIPTION** This routine maps PCMCIA memory onto a specified ISA address space.

**RETURNS** `OK`, or `ERROR` if the memory cannot be mapped.

**SEE ALSO** `sramDrv`

***srand()***

---

***srand()***

**NAME** *srand()* – reset the value of the seed used to generate random numbers (ANSI)

**SYNOPSIS**

```
void * srand
(
 uint_t seed /* random number seed */
)
```

**DESCRIPTION** This routine resets the seed value used by *rand()*. If *srand()* is then called with the same seed value, the sequence of pseudo-random numbers is repeated. If *rand()* is called before any calls to *srand()* have been made, the same sequence shall be generated as when *srand()* is first called with the seed value of 1.

**INCLUDE FILES** *stdlib.h*

**RETURNS** N/A

**SEE ALSO** *ansiStdlib, rand()*

---

***sscanf()***

**NAME** *sscanf()* – read and convert characters from an ASCII string (ANSI)

**SYNOPSIS**

```
int sscanf
(
 const char * str, /* string to scan */
 const char * fmt /* format string */
)
```

**DESCRIPTION** This routine reads characters from the string *str*, interprets them according to format specifications in the string *fmt*, which specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: one or more white-space

characters; an ordinary multibyte character (neither % nor a white-space character); or a conversion specification. Each conversion specification is introduced by the % character. After the %, the following appear in sequence:

- An optional assignment-suppressing character \*.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional **h** or **l** (**el**) indicating the size of the receiving object. The conversion specifiers **d**, **i**, and **n** should be preceded by **h** if the corresponding argument is a pointer to **short int** rather than a pointer to **int**, or by **l** if it is a pointer to **long int**. Similarly, the conversion specifiers **o**, **u**, and **x** shall be preceded by **h** if the corresponding argument is a pointer to **unsigned short int** rather than a pointer to **unsigned int**, or by **l** if it is a pointer to **unsigned long int**. Finally, the conversion specifiers **e**, **f**, and **g** shall be preceded by **l** if the corresponding argument is a pointer to **double** rather than a pointer to **float**. If an **h** or **l** appears with any other conversion specifier, the behavior is undefined.
- WARNING: ANSI C also specifies an optional **L** in some of the same contexts as **l** above, corresponding to a **long double \*** argument. However, the current release of the VxWorks libraries does not support **long double** data; using the optional **L** gives unpredictable results.
- A character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The **sscanf()** routine executes each directive of the format in turn. If a directive fails, as detailed below, **sscanf()** returns. Failures are described as input failures (due to the unavailability of input characters), or matching failures (due to inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary multibyte character is executed by reading the next characters of the stream. If one of the characters differs from one comprising the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

Input white-space characters (as specified by the **isspace()** function) are skipped, unless the specification includes a **l**, **c**, or **n** specifier.

An input item is read from the stream, unless the specification includes an **n** specifier. An input item is defined as the longest matching sequence of input characters, unless that exceeds a specified field width, in which case it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails: this condition is a matching

**sscanf()**

failure, unless an error prevented input from the stream, in which case it is an input failure.

Except in the case of a % specifier, the input item is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a \*, the result of the conversion is placed in the object pointed to by the first argument following the *fmt* argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The following conversion specifiers are valid:

**d**

Matches an optionally signed decimal integer whose format is the same as expected for the subject sequence of the *strtol()* function with the value 10 for the *base* argument. The corresponding argument should be a pointer to **int**.

**i**

Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the *strtol()* function with the value 0 for the *base* argument. The corresponding argument should be a pointer to **int**.

**o**

Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the *strtoul()* function with the value 8 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

**u**

Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtoul()* function with the value 10 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

**x**

Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the *strtoul()* function with the value 16 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

**e, f, g**

Match an optionally signed floating-point number, whose format is the same as expected for the subject string of the *strtod()* function. The corresponding argument should be a pointer to **float**.

**s**

Matches a sequence of non-white-space characters. The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which will be added automatically.

**[**

Matches a non-empty sequence of characters from a set of expected characters (the

**scanset**). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which is added automatically. The conversion specifier includes all subsequent character in the format string, up to and including the matching right bracket (]). The characters between the brackets (the **scanlist**) comprise the scanset, unless the character after the left bracket is a circumflex (^) in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with "[]" or "[^]", the right bracket character is in the scanlist and the next right bracket character is the matching right bracket that ends the specification; otherwise the first right bracket character is the one that ends the specification.

**c**

Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added.

**P**

Matches an implementation-defined set of sequences, which should be the same as the set of sequences that may be produced by the %p conversion of the *fprintf()* function. The corresponding argument should be a pointer to a pointer to **void**. VxWorks defines its pointer input field to be consistent with pointers written by the *fprintf()* function ("0x" hexadecimal notation). If the input item is a value converted earlier during the same program execution, the pointer that results should compare equal to that value; otherwise the behavior of the %p conversion is undefined.

**n**

No input is consumed. The corresponding argument should be a pointer to **int** into which the number of characters read from the input stream so far by this call to *sscanf()* is written. Execution of a %n directive does not increment the assignment count returned when *sscanf()* completes execution.

**%**

Matches a single %; no conversion or assignment occurs. The complete conversion specification is %%.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers **E**, **G**, and **X** are also valid and behave the same as **e**, **g**, and **x**, respectively.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the %n directive.

**INCLUDE FILES**    **fiolib.h**

**RETURNS**        The number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure; or EOF if an input failure occurs before any conversion.

**SEE ALSO**        **fiolib**, **fscanf()**, **scanf()**, *American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: Input/Output (stdio.h)*

---

## **st16552DevInit()**

**NAME**            **st16552DevInit()** – initialise an ST16552 channel

**SYNOPSIS**

```
void st16552DevInit
(
 ST16552_CHAN * pChan
)
```

**DESCRIPTION**    This routine initialises some SIO\_CHAN function pointers and then resets the chip in a quiescent state. Before this routine is called, the BSP must already have initialised all the device addresses, etc. in the ST16552\_CHAN structure.

**RETURNS**        N/A

**SEE ALSO**        **st16552Sio**

---

## **st16552Int()**

**NAME**            **st16552Int()** – interrupt level processing

**SYNOPSIS**

```
void st16552Int
(
 ST16552_CHAN * pChan /* ptr to struct describing channel */
)
```



**DESCRIPTION** This routine handles interrupts from the UART.

**RETURNS** N/A

**SEE ALSO** *st16552Sio*

---

## *st16552IntEx()*

**NAME** *st16552IntEx()* – miscellaneous interrupt processing

**SYNOPSIS**

```
void st16552IntEx
(
 ST16552_CHAN * pChan /* ptr to struct describing channel */
)
```

**DESCRIPTION** This routine handles miscellaneous interrupts on the UART.

**RETURNS** N/A

**SEE ALSO** *st16552Sio*

---

## *st16552IntRd()*

**NAME** *st16552IntRd()* – handle a receiver interrupt

**SYNOPSIS**

```
void st16552IntRd
(
 ST16552_CHAN * pChan /* ptr to struct describing channel */
)
```

**DESCRIPTION** This routine handles read interrupts from the UART.

**RETURNS** N/A

**SEE ALSO** *st16552Sio*

## ***st16552IntWr()***

**NAME** *st16552IntWr()* – handle a transmitter interrupt

**SYNOPSIS**

```
void st16552IntWr
(
 ST16552_CHAN * pChan /* ptr to struct describing channel */
)
```

**DESCRIPTION** This routine handles write interrupts from the UART.

**RETURNS** N/A

**SEE ALSO** *st16552Sio*

---

## ***st16552MuxInt()***

**NAME** *st16552MuxInt()* – multiplexed interrupt level processing

**SYNOPSIS**

```
void st16552MuxInt
(
 ST16552_MUX * pMux /* ptr to struct describing multiplexed chans */
)
```

**DESCRIPTION** This routine handles multiplexed interrupts from the DUART. It assumes that channels 0 and 1 are connected so that they produce the same interrupt.

**RETURNS** N/A

**SEE ALSO** *st16552Sio*

---

## *stat()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>stat()</i> – get file status information using a pathname (POSIX)                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS stat (     char *          name, /* name of file to check */     struct stat * pStat /* pointer to stat structure */ )</pre>                                                                                                                                                                                                                                                                    |
| <b>DESCRIPTION</b> | <p>This routine obtains various characteristics of a file (or directory). This routine is equivalent to <i>fstat()</i>, except that the <i>name</i> of the file is specified, rather than an open file descriptor.</p> <p>The <i>pStat</i> parameter is a pointer to a <b>stat</b> structure (defined in <b>stat.h</b>). This structure must have already been allocated before this routine is called.</p> |
| <b>NOTE</b>        | <p>When used with <b>netDrv</b> devices (FTP or RSH), <i>stat()</i> returns the size of the file and always sets the mode to regular; <i>stat()</i> does not distinguish between files, directories, links, etc.</p> <p>Upon return, the fields in the <b>stat</b> structure are updated to reflect the characteristics of the file.</p>                                                                    |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | <i>dirLib</i> , <i>fstat()</i> , <i>ls()</i>                                                                                                                                                                                                                                                                                                                                                                |

---

## *statfs()*

|                    |                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>statfs()</i> – get file status information using a pathname (POSIX)                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>STATUS statfs (     char *          name, /* name of file to check */     struct statfs * pStat /* pointer to statfs structure */ )</pre>                                                                   |
| <b>DESCRIPTION</b> | <p>This routine obtains various characteristics of a file system. This routine is equivalent to <i>fstatfs()</i>, except that the <i>name</i> of the file is specified, rather than an open file descriptor.</p> |

***stdioFp()***

The *pStat* parameter is a pointer to a **statfs** structure (defined in **stat.h**). This structure must have already been allocated before this routine is called.

Upon return, the fields in the **statfs** structure are updated to reflect the characteristics of the file.

**RETURNS** OK or ERROR.

**SEE ALSO** *dirLib*, *fstatfs()*, *ls()*

***stdioFp()***

**NAME** *stdioFp()* – return the standard input/output/error FILE of the current task

**SYNOPSIS**

```
FILE * stdioFp
(
 int stdFd /* fd of standard FILE to return (0,1,2) */
)
```

**DESCRIPTION** This routine returns the specified standard FILE structure address of the current task. It is provided primarily to give access to standard input, standard output, and standard error from the shell, where the usual **stdin**, **stdout**, **stderr** macros cannot be used.

**INCLUDE FILES** *stdio.h*

**RETURNS** The standard FILE structure address of the specified file descriptor, for the current task.

**SEE ALSO** *ansiStdio*

***stdioInit()***

**NAME** *stdioInit()* – initialize standard I/O support

**SYNOPSIS** `STATUS stdioInit (void)`

**DESCRIPTION** This routine installs standard I/O support. It must be called before using **stdio** buffering. If **INCLUDE\_STDIO** is defined in **configAll.h**, it is called automatically by the root task *usrRoot()* in *usrConfig.c*.

**RETURNS** OK, or ERROR if the standard I/O facilities cannot be installed.

**SEE ALSO** ansiStdio

---

## *stdioShow()*

**NAME** *stdioShow()* – display file pointer internals

**SYNOPSIS**

```
STATUS stdioShow
(
 FILE * fp, /* stream */
 int level /* level */
)
```

**DESCRIPTION** This routine displays information about a specified stream.

**RETURNS** OK, or ERROR if the file pointer is invalid.

**SEE ALSO** ansiStdio

---

## *stdioShowInit()*

**NAME** *stdioShowInit()* – initialize the standard I/O show facility

**SYNOPSIS** STATUS stdioShowInit (void)

**DESCRIPTION** This routine links the file pointer show routine into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define **INCLUDE\_SHOW\_ROUTINES** in **config.h**.
- If you use the Tornado project facility, select **INCLUDE\_STDIO\_SHOW**.

**RETURNS** OK, or ERROR if an error occurs installing the file pointer show routine.

**SEE ALSO** ansiStdio

**strcat()**

---

**strcat()**

|                      |                                                                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strcat()</i> – concatenate one string to another (ANSI)                                                                                                 |
| <b>SYNOPSIS</b>      | <pre>char * strcat (     char *      destination, /* string to be appended to */     const char * append     /* string to append to destination */ )</pre> |
| <b>DESCRIPTION</b>   | This routine appends a copy of string <i>append</i> to the end of string <i>destination</i> . The resulting string is null-terminated.                     |
| <b>INCLUDE FILES</b> | <code>string.h</code>                                                                                                                                      |
| <b>RETURNS</b>       | A pointer to <i>destination</i> .                                                                                                                          |
| <b>SEE ALSO</b>      | <code>ansiString</code>                                                                                                                                    |

---

**strchr()**

|                      |                                                                                                                                                 |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strchr()</i> – find the first occurrence of a character in a string (ANSI)                                                                   |
| <b>SYNOPSIS</b>      | <pre>char * strchr (     const char * s, /* string in which to search */     int         c  /* character to find in string */ )</pre>           |
| <b>DESCRIPTION</b>   | This routine finds the first occurrence of character <i>c</i> in string <i>s</i> . The terminating null is considered to be part of the string. |
| <b>INCLUDE FILES</b> | <code>string.h</code>                                                                                                                           |
| <b>RETURNS</b>       | The address of the located character, or NULL if the character is not found.                                                                    |
| <b>SEE ALSO</b>      | <code>ansiString</code>                                                                                                                         |

---

## *strcmp()*

|                      |                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strcmp()</i> – compare two strings lexicographically (ANSI)                                                                                                        |
| <b>SYNOPSIS</b>      | <pre>int strcmp (     const char * s1, /* string to compare */     const char * s2 /* string to compare s1 to */ )</pre>                                              |
| <b>DESCRIPTION</b>   | This routine compares string <i>s1</i> to string <i>s2</i> lexicographically.                                                                                         |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                       |
| <b>RETURNS</b>       | An integer greater than, equal to, or less than 0, according to whether <i>s1</i> is lexicographically greater than, equal to, or less than <i>s2</i> , respectively. |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                     |

---

## *strcoll()*

|                      |                                                                                                                                                                                                                      |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strcoll()</i> – compare two strings as appropriate to LC_COLLATE (ANSI)                                                                                                                                           |
| <b>SYNOPSIS</b>      | <pre>int strcoll (     const char * s1, /* string 1 */     const char * s2 /* string 2 */ )</pre>                                                                                                                    |
| <b>DESCRIPTION</b>   | This routine compares two strings, both interpreted as appropriate to the LC_COLLATE category of the current locale.                                                                                                 |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                                                                      |
| <b>RETURNS</b>       | An integer greater than, equal to, or less than zero, according to whether string <i>s1</i> is greater than, equal to, or less than string <i>s2</i> when both are interpreted as appropriate to the current locale. |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                                                                    |

## **strcpy()**

|                      |                                                                                                                        |
|----------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strcpy()</i> – copy one string to another (ANSI)                                                                    |
| <b>SYNOPSIS</b>      | <pre>char * strcpy (     char *      s1, /* string to copy to */     const char * s2 /* string to copy from */ )</pre> |
| <b>DESCRIPTION</b>   | This routine copies string <i>s2</i> (including EOS) to string <i>s1</i> .                                             |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                        |
| <b>RETURNS</b>       | A pointer to <i>s1</i> .                                                                                               |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                      |

---

## **strcspn()**

|                      |                                                                                                                                                             |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strcspn()</i> – return the string length up to the first character from a given set (ANSI)                                                               |
| <b>SYNOPSIS</b>      | <pre>size_t strcspn (     const char * s1, /* string to search */     const char * s2 /* set of characters to look for in s1 */ )</pre>                     |
| <b>DESCRIPTION</b>   | This routine computes the length of the maximum initial segment of string <i>s1</i> that consists entirely of characters not included in string <i>s2</i> . |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                             |
| <b>RETURNS</b>       | The length of the string segment.                                                                                                                           |
| <b>SEE ALSO</b>      | <b>ansiString, strpbrk(), strspn()</b>                                                                                                                      |



---

***strerror()***

**NAME** *strerror()* – map an error number to an error string (ANSI)

**SYNOPSIS**

```
char * strerror
(
 int errcode /* error code */
)
```

**DESCRIPTION** This routine maps the error number in *errcode* to an error message string. It returns a pointer to a static buffer that holds the error string.

This routine is not reentrant. For a reentrant version, see *strerror\_r()*.

**INCLUDE** *string.h*

**RETURNS** A pointer to the buffer that holds the error string.

**SEE ALSO** *ansiString, strerror\_r()*

---

***strerror\_r()***

**NAME** *strerror\_r()* – map an error number to an error string (POSIX)

**SYNOPSIS**

```
STATUS strerror_r
(
 int errcode, /* error code */
 char * buffer /* string buffer */
)
```

**DESCRIPTION** This routine maps the error number in *errcode* to an error message string. It stores the error string in *buffer*.

This routine is the POSIX reentrant version of *strerror()*.

**INCLUDE FILES** *string.h*

**RETURNS** OK or ERROR.

**SEE ALSO** *ansiString, strerror()*

---

## **strftime()**

**NAME** `strftime()` – convert broken-down time into a formatted string (ANSI)

**SYNOPSIS**

```
size_t strftime
(
 char * s, /* string array */
 size_t n, /* maximum size of array */
 const char * format, /* format of output string */
 const struct tm * tptr /* broken-down time */
)
```

**DESCRIPTION** This routine formats the broken-down time in *tptr* based on the conversion specified in the string *format*, and places the result in the string *s*.

The format is a multibyte character sequence, beginning and ending in its initial state. The *format* string consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a % character followed by a character that determines the behavior of the conversion. All ordinary multibyte characters (including the terminating NULL character) are copied unchanged to the array. If copying takes place between objects that overlap, the behavior is undefined. No more than *n* characters are placed into the array.

Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the LC\_TIME category of the current locale and by the values contained in the structure pointed to by *tptr*.

- %a the locale's abbreviated weekday name.
- %A the locale's full weekday name.
- %b the locale's abbreviated month name.
- %B the locale's full month name.
- %c the locale's appropriate date and time representation.
- %d the day of the month as decimal number (01-31).
- %H the hour (24-hour clock) as a decimal number (00-23).
- %I the hour (12-hour clock) as a decimal number (01-12).
- %j the day of the year as decimal number (001-366).
- %m the month as a decimal number (01-12).
- %M the minute as a decimal number (00-59).
- %P the locale's equivalent of the AM/PM designations associated with a 12-hour clock.

|                 |                                                                                                        |
|-----------------|--------------------------------------------------------------------------------------------------------|
| <code>%S</code> | the second as a decimal number (00-59).                                                                |
| <code>%U</code> | the week number of the year (first Sunday as the first day of week 1) as a decimal number (00-53).     |
| <code>%w</code> | the weekday as a decimal number (0-6), where Sunday is 0.                                              |
| <code>%W</code> | the week number of the year (the first Monday as the first day of week 1) as a decimal number (00-53). |
| <code>%x</code> | the locale's appropriate date representation.                                                          |
| <code>%X</code> | the locale's appropriate time representation.                                                          |
| <code>%y</code> | the year without century as a decimal number (00-99).                                                  |
| <code>%Y</code> | the year with century as a decimal number.                                                             |
| <code>%Z</code> | the time zone name or abbreviation, or by no characters if no time zone is determinable.               |
| <code>%%</code> | <code>%</code> .                                                                                       |

**INCLUDE FILES**     `time.h`

**RETURNS**            The number of characters in *s*, not including the terminating null character -- or zero if the number of characters in *s*, including the null character, is more than *n* (in which case the contents of *s* are indeterminate).

**SEE ALSO**            `ansiTime`

---

## ***strlen()***

**NAME**                `strlen()` – determine the length of a string (ANSI)

**SYNOPSIS**            `size_t strlen`  
                      (  
                      `const char * s /* string */`  
                      )

**DESCRIPTION**        This routine returns the number of characters in *s*, not including EOS.

**INCLUDE FILES**     `string.h`

**RETURNS**            The number of non-null characters in the string.

**SEE ALSO**            `ansiString`

**strncat()**

---

**strncat()**

**NAME** *strncat()* – concatenate characters from one string to another (ANSI)

**SYNOPSIS**

```
char * strncat
(
 char * dst, /* string to append to */
 const char * src, /* string to append */
 size_t n /* max no. of characters to append */
)
```

**DESCRIPTION** This routine appends up to *n* characters from string *src* to the end of string *dst*.

**INCLUDE FILES** `string.h`

**RETURNS** A pointer to the null-terminated string *s1*.

**SEE ALSO** `ansiString`

---

**strncmp()**

**NAME** *strncmp()* – compare the first *n* characters of two strings (ANSI)

**SYNOPSIS**

```
int strncmp
(
 const char * s1, /* string to compare */
 const char * s2, /* string to compare s1 to */
 size_t n /* max no. of characters to compare */
)
```

**DESCRIPTION** This routine compares up to *n* characters of string *s1* to string *s2*lexicographically.

**INCLUDE FILES** `string.h`

**RETURNS** An integer greater than, equal to, or less than 0, according to whether *s1* is lexicographically greater than, equal to, or less than *s2*, respectively.

**SEE ALSO** `ansiString`

---

## *strncpy()*

|                      |                                                                                                                                                                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strncpy()</i> – copy characters from one string to another (ANSI)                                                                                                                                                                                                                   |
| <b>SYNOPSIS</b>      | <pre>char *strncpy (     char *      s1, /* string to copy to */     const char * s2, /* string to copy from */     size_t      n /* max no. of characters to copy */ )</pre>                                                                                                          |
| <b>DESCRIPTION</b>   | This routine copies <i>n</i> characters from string <i>s2</i> to string <i>s1</i> . If <i>n</i> is greater than the length of <i>s2</i> , nulls are added to <i>s1</i> . If <i>n</i> is less than or equal to the length of <i>s2</i> , the target string will not be null-terminated. |
| <b>INCLUDE FILES</b> | <i>string.h</i>                                                                                                                                                                                                                                                                        |
| <b>RETURNS</b>       | A pointer to <i>s1</i> .                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>      | <i>ansiString</i>                                                                                                                                                                                                                                                                      |

---

## *strpbrk()*

|                      |                                                                                                                                         |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strpbrk()</i> – find the first occurrence in a string of a character from a given set (ANSI)                                         |
| <b>SYNOPSIS</b>      | <pre>char * strpbrk (     const char * s1, /* string to search */     const char * s2 /* set of characters to look for in s1 */ )</pre> |
| <b>DESCRIPTION</b>   | This routine locates the first occurrence in string <i>s1</i> of any character from string <i>s2</i> .                                  |
| <b>INCLUDE FILES</b> | <i>string.h</i>                                                                                                                         |
| <b>RETURNS</b>       | A pointer to the character found in <i>s1</i> , or NULL if no character from <i>s2</i> occurs in <i>s1</i> .                            |
| <b>SEE ALSO</b>      | <i>ansiString</i> , <i>strcspn()</i>                                                                                                    |

## ***strrchr()***

**NAME** *strrchr()* – find the last occurrence of a character in a string (ANSI)

**SYNOPSIS**

```
char * strrchr
(
 const char * s, /* string to search */
 int c /* character to look for */
)
```

**DESCRIPTION** This routine locates the last occurrence of *c* in the string pointed to by *s*. The terminating null is considered to be part of the string.

**INCLUDE FILES** *string.h*

**RETURNS** A pointer to the last occurrence of the character, or NULL if the character is not found.

**SEE ALSO** *ansiString*

---

## ***strspn()***

**NAME** *strspn()* – return the string length up to the first character not in a given set (ANSI)

**SYNOPSIS**

```
size_t strspn
(
 const char * s, /* string to search */
 const char * sep /* set of characters to look for in s */
)
```

**DESCRIPTION** This routine computes the length of the maximum initial segment of string *s* that consists entirely of characters from the string *sep*.

**INCLUDE FILES** *string.h*

**RETURNS** The length of the string segment.

**SEE ALSO** *ansiString, strcspn()*

---

## ***strstr()***

|                      |                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strstr()</i> – find the first occurrence of a substring in a string (ANSI)                                                                                     |
| <b>SYNOPSIS</b>      | <pre>char * strstr (     const char * s, /* string to search */     const char * find /* substring to look for */ )</pre>                                         |
| <b>DESCRIPTION</b>   | This routine locates the first occurrence in string <i>s</i> of the sequence of characters (excluding the terminating null character) in the string <i>find</i> . |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                   |
| <b>RETURNS</b>       | A pointer to the located substring, or <i>s</i> if <i>find</i> points to a zero-length string, or NULL if the string is not found.                                |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                 |

---

## ***strtod()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>strtod()</i> – convert the initial portion of a string to a double (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>double strtod (     const char * s, /* string to convert */     char * * endptr /* ptr to final string */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | <p>This routine converts the initial portion of a specified string <i>s</i> to a double. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by the <i>isspace()</i> function); a subject sequence resembling a floating-point constant; and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, it attempts to convert the subject sequence to a floating-point number, and returns the result.</p> <p>The expected form of the subject sequence is an optional plus or minus decimal-point character, then an optional exponent part but no floating suffix. The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no</p> |

***strtok()***

characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign, a digit, or a decimal-point character.

If the subject sequence has the expected form, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) is interpreted as a floating constant, except that the decimal-point character is used in place of a period, and that if neither an exponent part nor a decimal-point character appears, a decimal point is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the "C" locale, additional implementation-defined subject sequence forms may be accepted. VxWorks supports only the "C" locale.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *s* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

**INCLUDE FILES**     **stdlib.h**

**RETURNS**           The converted value, if any. If no conversion could be performed, it returns zero. If the correct value is outside the range of representable values, it returns plus or minus **HUGE\_VAL** (according to the sign of the value), and stores the value of the macro **ERANGE** in **errno**. If the correct value would cause underflow, it returns zero and stores the value of the macro **ERANGE** in **errno**.

**SEE ALSO**           **ansiStdlib**

---

## ***strtok()***

**NAME**               ***strtok()*** – break down a string into tokens (ANSI)

**SYNOPSIS**

```
char * strtok
(
 char * string, /* string */
 const char * separator /* separator indicator */
)
```

**DESCRIPTION**     A sequence of calls to this routine breaks the string *string* into a sequence of tokens, each of which is delimited by a character from the string *separator*. The first call in the sequence has *string* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string may be different from call to call.



The first call in the sequence searches *string* for the first character that is not contained in the current separator string. If the character is not found, there are no tokens in *string* and *strtok()* returns a null pointer. If the character is found, it is the start of the first token.

*strtok()* then searches from there for a character that is contained in the current separator string. If the character is not found, the current token expands to the end of the string pointed to by *string*, and subsequent searches for a token will return a null pointer. If the character is found, it is overwritten by a null character, which terminates the current token. *strtok()* saves a pointer to the following character, from which the next search for a token will start. (Note that because the separator character is overwritten by a null character, the input string is modified as a result of this call.)

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

The implementation behaves as if *strtok()* is called by no library functions.

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| REENTRANCY    | This routine is not reentrant; the reentrant form is <i>strtok_r()</i> .             |
| INCLUDE FILES | <i>string.h</i>                                                                      |
| RETURNS       | A pointer to the first character of a token, or a NULL pointer if there is no token. |
| SEE ALSO      | <i>ansiString</i> , <i>strtok_r()</i>                                                |

---

## *strtok\_r()*

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NAME        | <i>strtok_r()</i> – break down a string into tokens (reentrant) (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| SYNOPSIS    | <pre>char * strtok_r (     char *      string,      /* string to break into tokens */     const char * separators, /* the separators */     char * *    ppLast      /* pointer to serve as string index */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| DESCRIPTION | <p>This routine considers the null-terminated string <i>string</i> as a sequence of zero or more text tokens separated by spans of one or more characters from the separator string <i>separators</i>. The argument <i>ppLast</i> points to a user-provided pointer which in turn points to the position within <i>string</i> at which scanning should begin.</p> <p>In the first call to this routine, <i>string</i> points to a null-terminated string; <i>separators</i> points to a null-terminated string of separator characters; and <i>ppLast</i> points to a NULL pointer. The function returns a pointer to the first character of the first token, writes a null character</p> |

**strtol()**

into *string* immediately following the returned token, and updates the pointer to which *ppLast* points so that it points to the first character following the null written into *string*. (Note that because the separator character is overwritten by a null character, the input string is modified as a result of this call.)

In subsequent calls *string* must be a NULL pointer and *ppLast* must be unchanged so that subsequent calls will move through the string *string*, returning successive tokens until no tokens remain. The separator string *separators* may be different from call to call. When no token remains in *string*, a NULL pointer is returned.

**INCLUDE FILES**     **string.h**

**RETURNS**            A pointer to the first character of a token, or a NULL pointer if there is no token.

**SEE ALSO**            **ansiString, strtok()**

---

## **strtol()**

**NAME**                **strtol()** – convert a string to a long integer (ANSI)

**SYNOPSIS**

```
long strtol
(
 const char * nptr, /* string to convert */
 char * * endptr, /* ptr to final string */
 int base /* radix */
)
```

**DESCRIPTION**       This routine converts the initial portion of a string *nptr* to **long int** representation. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by *isspace()*); a subject sequence resembling an integer represented in some radix determined by the value of *base*; and a final string of one or more unrecognized characters, including the terminating NULL character of the input string. Then, it attempts to convert the subject sequence to an integer number, and returns the result.

If the value of *base* is zero, the expected form of the subject sequence is that of an integer constant, optionally preceded by a plus or minus sign, but not including an integer suffix. If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base* optionally preceded by a plus or minus sign, but not including an integer suffix. The letters from a (or A) through to z (or Z) are ascribed the values 10 to 35; only letters whose ascribed values are less than *base* are premitted. If the value of *base* is 16, the characters 0x

or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the *base* for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a NULL pointer.

In other than the "C" locale, additional implementation-defined subject sequence forms may be accepted. VxWorks supports only the "C" locale; it assumes that the upper- and lower-case alphabets and digits are each contiguous.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a NULL pointer.

**INCLUDE FILES**     **stdlib.h**

**RETURNS**           The converted value, if any. If no conversion could be performed, it returns zero. If the correct value is outside the range of representable values, it returns **LONG\_MAX** or **LONG\_MIN** (according to the sign of the value), and stores the value of the macro **ERANGE** in **errno**.

**SEE ALSO**           **ansiStdlib**

---

## **strtoul()**

**NAME**               **strtoul()** – convert a string to an unsigned long integer (ANSI)

**SYNOPSIS**           **ulong\_t strtoul**  
                  (  
                  **const char \* nptr,**    /\* string to convert \*/  
                  **char \* \*        endptr,** /\* ptr to final string \*/  
                  **int             base    /\* radix \*/**  
                  )

**strtoul()****DESCRIPTION**

This routine converts the initial portion of a string *nptr* to **unsigned long int** representation. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by *isspace()*); a subject sequence resembling an unsigned integer represented in some radix determined by the value *base*; and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, it attempts to convert the subject sequence to an unsigned integer, and returns the result.

If the value of *base* is zero, the expected form of the subject sequence is that of an integer constant, optionally preceded by a plus or minus sign, but not including an integer suffix. If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by letters from a (or A) through z (or Z) which are ascribed the values 10 to 35; only letters whose ascribed values are less than *base* are premitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the *base* for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the "C" locale, additional implementation-defined subject sequence forms may be accepted. VxWorks supports only the "C" locale; it assumes that the upper- and lower-case alphabets and digits are each contiguous.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

**INCLUDE FILES****stdlib.h****RETURNS**

The converted value, if any. If no conversion could be performed it returns zero. If the correct value is outside the range of representable values, it returns **ULONG\_MAX**, and stores the value of the macro **ERANGE** in *errno*.

**SEE ALSO****ansiStdlib**

---

## *strxfrm()*

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>strxfrm()</i> – transform up to <i>n</i> characters of <i>s2</i> into <i>s1</i> (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>      | <pre>size_t strxfrm (     char *      s1, /* string out */     const char * s2, /* string in */     size_t     n /* size of buffer */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b>   | This routine transforms string <i>s2</i> and places the resulting string in <i>s1</i> . The transformation is such that if <i>strcmp()</i> is applied to two transformed strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the <i>strcoll()</i> function applied to the same two original strings. No more than <i>n</i> characters are placed into the resulting <i>s1</i> , including the terminating null character. If <i>n</i> is zero, <i>s1</i> is permitted to be a NULL pointer. If copying takes place between objects that overlap, the behavior is undefined. |
| <b>INCLUDE FILES</b> | <i>string.h</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>RETURNS</b>       | The length of the transformed string, not including the terminating null character. If the value is <i>n</i> or more, the contents of <i>s1</i> are indeterminate.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>      | <i>ansiString</i> , <i>strcmp()</i> , <i>strcoll()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

---

## *swab()*

|                    |                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>swab()</i> – swap bytes                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>void swab (     char * source,      /* pointer to source buffer */     char * destination, /* pointer to destination buffer */     int  nbytes        /* number of bytes to exchange */ )</pre>                       |
| <b>DESCRIPTION</b> | This routine gets the specified number of bytes from <i>source</i> , exchanges the adjacent even and odd bytes, and puts them in <i>destination</i> . The buffers <i>source</i> and <i>destination</i> should not overlap. |

**symAdd()**

|                 |                                                                                                                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NOTE</b>     | On some CPUs, <i>swab()</i> will cause an exception if the buffers are unaligned. In such cases, use <i>uswab()</i> for unaligned swaps.<br><br>It is an error for <i>nbytes</i> to be odd. |
| <b>RETURNS</b>  | N/A                                                                                                                                                                                         |
| <b>SEE ALSO</b> | <b>bLib</b> , <i>uswab()</i>                                                                                                                                                                |

---

**symAdd()**

|                    |                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>symAdd()</i> – create and add a symbol to a symbol table, including a group number                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre> STATUS symAdd (     SYMTAB_ID symTblId, /* symbol table to add symbol to */     char *    name,     /* pointer to symbol name string */     char *    value,    /* symbol address */     SYM_TYPE  type,     /* symbol type */     UINT16    group     /* symbol group */ ) </pre>                                    |
| <b>DESCRIPTION</b> | This routine allocates a symbol <i>name</i> and adds it to a specified symbol table <i>symTblId</i> with the specified parameters <i>value</i> , <i>type</i> , and <i>group</i> . The <i>group</i> parameter specifies the group number assigned to a module when it is loaded; see the manual entry for <b>moduleLib</b> . |
| <b>RETURNS</b>     | OK, or ERROR if the symbol table is invalid or there is insufficient memory for the symbol to be allocated.                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>symLib</b> , <b>moduleLib</b>                                                                                                                                                                                                                                                                                            |

---

**symEach()**

|                 |                                                                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <i>symEach()</i> – call a routine to examine each entry in a symbol table                                                                            |
| <b>SYNOPSIS</b> | <pre> SYMBOL *symEach (     SYMTAB_ID symTblId, /* pointer to symbol table */     FUNCPTR   routine,  /* func to call for each tbl entry */ ) </pre> |

```

int routineArg /* arbitrary user-supplied arg */
)

```

**DESCRIPTION** This routine calls a user-supplied routine to examine each entry in the symbol table; it calls the specified routine once for each entry. The routine should be declared as follows:

```

BOOL routine
(
char *name, /* entry name */
int val, /* value associated with entry */
SYM_TYPE type, /* entry type */
int arg, /* arbitrary user-supplied arg */
UINT16 group /* group number */
)

```

The user-supplied routine should return TRUE if *symEach()* is to continue calling it for each entry, or FALSE if it is done and *symEach()* can exit.

**RETURNS** A pointer to the last symbol reached, or NULL if all symbols are reached.

**SEE ALSO** *symLib*

---

## *symFindByName()*

**NAME** *symFindByName()* – look up a symbol by name

**SYNOPSIS**

```

STATUS symFindByName
(
SYM_TAB_ID symTblId, /* ID of symbol table to look in */
char * name, /* symbol name to look for */
char * *pValue, /* where to put symbol value */
SYM_TYPE * pType /* where to put symbol type */
)

```

**DESCRIPTION** This routine searches a symbol table for a symbol matching a specified name. If the symbol is found, its value and type are copied to *pValue* and *pType*. If multiple symbols have the same name but differ in type, the routine chooses the matching symbol most recently added to the symbol table.

To search the global VxWorks symbol table, specify *sysSymTbl* as *symTblId*.

**RETURNS** OK, or ERROR if the symbol table ID is invalid or the symbol cannot be found.

**SEE ALSO** *symLib*

---

## **symFindByNameAndType()**

**NAME** *symFindByNameAndType()* – look up a symbol by name and type

**SYNOPSIS**

```
STATUS symFindByNameAndType
(
 SYMTAB_ID symTblId, /* ID of symbol table to look in */
 char * name, /* symbol name to look for */
 char * *pValue, /* where to put symbol value */
 SYM_TYPE * pType, /* where to put symbol type */
 SYM_TYPE sType, /* symbol type to look for */
 SYM_TYPE mask /* bits in sType to pay attention to */
)
```

**DESCRIPTION** This routine searches a symbol table for a symbol matching both name and type (*name* and *sType*). If the symbol is found, its value and type are copied to *pValue* and *pType*. The *mask* parameter can be used to match sub-classes of type.

To search the global VxWorks symbol table, specify **sysSymTbl** as *symTblId*.

**RETURNS** OK, or ERROR if the symbol table ID is invalid or the symbol is not found.

**SEE ALSO** *symLib*

---

## **symFindByValue()**

**NAME** *symFindByValue()* – look up a symbol by value

**SYNOPSIS**

```
STATUS symFindByValue
(
 SYMTAB_ID symTblId, /* ID of symbol table to look in */
 UINT value, /* value of symbol to find */
 char * name, /* where to put symbol name string */
 int * pValue, /* where to put symbol value */
 SYM_TYPE * pType /* where to put symbol type */
)
```

**DESCRIPTION** This routine searches a symbol table for a symbol matching a specified value. If there is no matching entry, it chooses the table entry with the next lower value. The symbol name (with terminating EOS), the actual value, and the type are copied to *name*, *pValue*, and *pType*.



For the *name* buffer, allocate `MAX_SYS_SYM_LEN + 1` bytes. The value `MAX_SYS_SYM_LEN` is defined in `sysSymTbl.h`.

To search the global VxWorks symbol table, specify `sysSymTbl` as *symTblId*.

**RETURNS** OK, or ERROR if *value* is less than the lowest value in the table.

**SEE ALSO** `symLib`

## *symFindByValueAndType()*

**NAME** *symFindByValueAndType()* – look up a symbol by value and type

**SYNOPSIS**

```
STATUS symFindByValueAndType
(
 SYMTAB_ID symTblId, /* ID of symbol table to look in */
 UINT value, /* value of symbol to find */
 char * name, /* where to put symbol name string */
 int * pValue, /* where to put symbol value */
 SYM_TYPE * pType, /* where to put symbol type */
 SYM_TYPE sType, /* symbol type to look for */
 SYM_TYPE mask /* bits in sType to pay attention to */
)
```

**DESCRIPTION** This routine searches a symbol table for a symbol matching both value and type (*value* and *sType*). If there is no matching entry, it chooses the table entry with the next lower value. The symbol name (with terminating EOS), the actual value, and the type are copied to *name*, *pValue*, and *pType*. The *mask* parameter can be used to match sub-classes of type.

For the *name* buffer, allocate `MAX_SYS_SYM_LEN + 1` bytes. The value `MAX_SYS_SYM_LEN` is defined in `sysSymTbl.h`.

To search the global VxWorks symbol table, specify `sysSymTbl` as *symTblId*.

**RETURNS** OK, or ERROR if *value* is less than the lowest value in the table.

**SEE ALSO** `symLib`

## ***symLibInit()***

**NAME** *symLibInit()* – initialize the symbol table library

**SYNOPSIS** `STATUS symLibInit (void)`

**DESCRIPTION** This routine initializes the symbol table package. If the configuration macro `INCLUDE_SYM_TBL` is defined, *symLibInit()* is called by the root task, *usrRoot()*, in `usrConfig.c`.

**RETURNS** OK, or ERROR if the library could not be initialized.

**SEE ALSO** `symLib`

---

## ***symRemove()***

**NAME** *symRemove()* – remove a symbol from a symbol table

**SYNOPSIS** `STATUS symRemove`

```
(
 SYMTAB_ID symTblId, /* symbol tbl to remove symbol from */
 char * name, /* name of symbol to remove */
 SYM_TYPE type /* type of symbol to remove */
)
```

**DESCRIPTION** This routine removes a symbol of matching name and type from a specified symbol table. The symbol is deallocated if found. Note that VxWorks symbols in a standalone VxWorks image (where the symbol table is linked in) cannot be removed.

**RETURNS** OK, or ERROR if the symbol is not found or could not be deallocated.

**SEE ALSO** `symLib`

---

## *symSyncLibInit()*

|                    |                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>symSyncLibInit()</i> – initialize host/target symbol table synchronization                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>void symSyncLibInit ( )</pre>                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | This routine initializes host/target symbol table synchronization. To enable synchronization, it must be called before a target server is started. It is called automatically if the configuration macro <code>INCLUDE_SYM_TBL_SYNC</code> is defined. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <code>symSyncLib</code>                                                                                                                                                                                                                                |

---

## *symSyncTimeoutSet()*

|                    |                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>symSyncTimeoutSet()</i> – set WTX timeout                                                             |
| <b>SYNOPSIS</b>    | <pre>UINT32 symSyncTimeoutSet<br/>(<br/>    UINT32 timeout /* WTX timeout in milliseconds */<br/>)</pre> |
| <b>DESCRIPTION</b> | This routine sets the WTX timeout between target server and synchronization task.                        |
| <b>RETURNS</b>     | If <i>timeout</i> is 0, the current timeout, otherwise the new timeout value in milliseconds.            |
| <b>SEE ALSO</b>    | <code>symSyncLib</code>                                                                                  |

---

## *symTblCreate()*

|                 |                                                                                                                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <i>symTblCreate()</i> – create a symbol table                                                                                                                                             |
| <b>SYNOPSIS</b> | <pre>SYMTAB_ID symTblCreate<br/>(<br/>    int     hashSizeLog2, /* size of hash table as a power of 2 */<br/>    BOOL    sameNameOk,  /* allow 2 symbols of same name &amp; type */</pre> |

***symTblDelete()***

```

 PART_ID symPartId /* memory part ID for symbol allocation */
)

```

- DESCRIPTION** This routine creates and initializes a symbol table with a hash table of a specified size. The size of the hash table is specified as a power of two. For example, if *hashSizeLog2* is 6, a 64-entry hash table is created.
- If *sameNameOk* is FALSE, attempting to add a symbol with the same name and type as an already-existing symbol results in an error.
- Memory for storing symbols as they are added to the symbol table will be allocated from the memory partition *symPartId*. The ID of the system memory partition is stored in the global variable **memSysPartId**, which is declared in **memLib.h**.
- RETURNS** Symbol table ID, or NULL if memory is insufficient.
- SEE ALSO** **symLib**

---

## ***symTblDelete()***

- NAME** *symTblDelete()* – delete a symbol table
- SYNOPSIS**
- ```

STATUS symTblDelete
(
    SYMTAB_ID symTblId /* ID of symbol table to delete */
)

```
- DESCRIPTION** This routine deletes a specified symbol table. It deallocates all associated memory, including the hash table, and marks the table as invalid.
- Deletion of a table that still contains symbols results in ERROR. Successful deletion includes the deletion of the internal hash table and the deallocation of memory associated with the table. The table is marked invalid to prohibit any future references.
- RETURNS** OK, or ERROR if the symbol table ID is invalid.
- SEE ALSO** **symLib**

sysAuxClkConnect()

| | |
|--------------------|--|
| NAME | <i>sysAuxClkConnect()</i> – connect a routine to the auxiliary clock interrupt |
| SYNOPSIS | <pre>STATUS sysAuxClkConnect (FUNCPTR routine, /* routine called at each aux clock interrupt */ int arg /* argument to auxiliary clock interrupt routine */)</pre> |
| DESCRIPTION | This routine specifies the interrupt service routine to be called at each auxiliary clock interrupt. It does not enable auxiliary clock interrupts. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | OK, or ERROR if the routine cannot be connected to the interrupt. |
| SEE ALSO | <i>sysLib</i> , <i>intConnect()</i> , <i>sysAuxClkEnable()</i> , and BSP-specific reference pages for this routine |

sysAuxClkDisable()

| | |
|--------------------|--|
| NAME | <i>sysAuxClkDisable()</i> – turn off auxiliary clock interrupts |
| SYNOPSIS | <pre>void sysAuxClkDisable (void)</pre> |
| DESCRIPTION | This routine disables auxiliary clock interrupts. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | N/A |
| SEE ALSO | <i>sysLib</i> , <i>sysAuxClkEnable()</i> , and BSP-specific reference pages for this routine |

sysAuxClkEnable()

| | |
|--------------------|--|
| NAME | <i>sysAuxClkEnable()</i> – turn on auxiliary clock interrupts |
| SYNOPSIS | <code>void sysAuxClkEnable (void)</code> |
| DESCRIPTION | This routine enables auxiliary clock interrupts. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | N/A |
| SEE ALSO | <i>sysLib</i> , <i>sysAuxClkConnect()</i> , <i>sysAuxClkDisable()</i> , <i>sysAuxClkRateSet()</i> , and BSP-specific reference pages for this routine |

sysAuxClkRateGet()

| | |
|--------------------|--|
| NAME | <i>sysAuxClkRateGet()</i> – get the auxiliary clock rate |
| SYNOPSIS | <code>int sysAuxClkRateGet (void)</code> |
| DESCRIPTION | This routine returns the interrupt rate of the auxiliary clock. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | The number of ticks per second of the auxiliary clock. |
| SEE ALSO | <i>sysLib</i> , <i>sysAuxClkEnable()</i> , <i>sysAuxClkRateSet()</i> , and BSP-specific reference pages for this routine |

sysAuxClkRateSet()

| | |
|--------------------|--|
| NAME | <i>sysAuxClkRateSet()</i> – set the auxiliary clock rate |
| SYNOPSIS | <pre>STATUS sysAuxClkRateSet (int ticksPerSecond /* number of clock interrupts per second */)</pre> |
| DESCRIPTION | This routine sets the interrupt rate of the auxiliary clock. It does not enable auxiliary clock interrupts. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | OK, or ERROR if the tick rate is invalid or the timer cannot be set. |
| SEE ALSO | <i>sysLib</i> , <i>sysAuxClkEnable()</i> , <i>sysAuxClkRateGet()</i> , and BSP-specific reference pages for this routine |

sysBspRev()

| | |
|--------------------|--|
| NAME | <i>sysBspRev()</i> – return the BSP version and revision number |
| SYNOPSIS | <pre>char * sysBspRev (void)</pre> |
| DESCRIPTION | This routine returns a pointer to a BSP version and revision number, for example, 1.0/1. BSP_REV is concatenated to BSP_VERSION and returned. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | A pointer to the BSP version/revision string. |
| SEE ALSO | <i>sysLib</i> , and BSP-specific reference pages for this routine |

sysBusIntAck()

| | |
|--------------------|---|
| NAME | <i>sysBusIntAck()</i> – acknowledge a bus interrupt |
| SYNOPSIS | <pre>int sysBusIntAck (int intLevel /* interrupt level to acknowledge */)</pre> |
| DESCRIPTION | This routine acknowledges a specified VMEbus interrupt level. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.* |
| RETURNS | NULL. |
| SEE ALSO | <i>sysLib</i> , <i>sysBusIntGen()</i> , and BSP-specific reference pages for this routine |

sysBusIntGen()

| | |
|--------------------|---|
| NAME | <i>sysBusIntGen()</i> – generate a bus interrupt |
| SYNOPSIS | <pre>STATUS sysBusIntGen (int intLevel, /* bus interrupt level to generate */ int vector /* interrupt vector to generate (0-255) */)</pre> |
| DESCRIPTION | This routine generates a bus interrupt for a specified level with a specified vector. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.* |
| RETURNS | OK, or ERROR if <i>intLevel</i> is out of range or the board cannot generate a bus interrupt. |
| SEE ALSO | <i>sysLib</i> , <i>sysBusIntAck()</i> , and BSP-specific reference pages for this routine |

sysBusTas()

| | |
|--------------------|---|
| NAME | <i>sysBusTas()</i> – test and set a location across the bus |
| SYNOPSIS | <pre>BOOL sysBusTas (char * adrs /* address to be tested and set */)</pre> |
| DESCRIPTION | This routine performs a test-and-set instruction across the backplane. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| NOTE | This routine is equivalent to <i>vxTas()</i> . |
| RETURNS | TRUE if the value had not been set but is now, or FALSE if the value was set already. |
| SEE ALSO | <i>sysLib</i> , <i>vxTas()</i> , and BSP-specific reference pages for this routine |

sysBusToLocalAdrs()

| | |
|--------------------|---|
| NAME | <i>sysBusToLocalAdrs()</i> – convert a bus address to a local address |
| SYNOPSIS | <pre>STATUS sysBusToLocalAdrs (int adrsSpace, /* bus address space in which busAdrs resides */ char * busAdrs, /* bus address to convert */ char * *pLocalAdrs /* where to return local address */)</pre> |
| DESCRIPTION | This routine gets the local address that accesses a specified bus memory address. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | OK, or ERROR if the address space is unknown or the mapping is not possible. |
| SEE ALSO | <i>sysLib</i> , <i>sysLocalToBusAdrs()</i> , and BSP-specific reference pages for this routine |

sysClkConnect()

| | |
|--------------------|--|
| NAME | <i>sysClkConnect()</i> – connect a routine to the system clock interrupt |
| SYNOPSIS | <pre>STATUS sysClkConnect (FUNCPTR routine, /* routine called at each system clock interrupt */ int arg /* argument with which to call routine */)</pre> |
| DESCRIPTION | This routine specifies the interrupt service routine to be called at each clock interrupt. Normally, it is called from <i>usrRoot()</i> in <i>usrConfig.c</i> to connect <i>usrClock()</i> to the system clock interrupt. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURN | OK, or ERROR if the routine cannot be connected to the interrupt. |
| SEE ALSO | <i>sysLib</i> , <i>intConnect()</i> , <i>usrClock()</i> , <i>sysClkEnable()</i> , and BSP-specific reference pages for this routine |

sysClkDisable()

| | |
|--------------------|--|
| NAME | <i>sysClkDisable()</i> – turn off system clock interrupts |
| SYNOPSIS | <pre>void sysClkDisable (void)</pre> |
| DESCRIPTION | This routine disables system clock interrupts. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | N/A |
| SEE ALSO | <i>sysLib</i> , <i>sysClkEnable()</i> , and BSP-specific reference pages for this routine |

sysClkEnable()

| | |
|--------------------|--|
| NAME | <i>sysClkEnable()</i> – turn on system clock interrupts |
| SYNOPSIS | <code>void sysClkEnable (void)</code> |
| DESCRIPTION | This routine enables system clock interrupts. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | N/A |
| SEE ALSO | <i>sysLib</i> , <i>sysClkConnect()</i> , <i>sysClkDisable()</i> , <i>sysClkRateSet()</i> , and BSP-specific reference pages for this routine |

sysClkRateGet()

| | |
|--------------------|--|
| NAME | <i>sysClkRateGet()</i> – get the system clock rate |
| SYNOPSIS | <code>int sysClkRateGet (void)</code> |
| DESCRIPTION | This routine returns the system clock rate. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | The number of ticks per second of the system clock. |
| SEE ALSO | <i>sysLib</i> , <i>sysClkEnable()</i> , <i>sysClkRateSet()</i> , and BSP-specific reference pages for this routine |

sysClkRateSet()

| | |
|--------------------|--|
| NAME | <i>sysClkRateSet()</i> – set the system clock rate |
| SYNOPSIS | <pre>STATUS sysClkRateSet (int ticksPerSecond /* number of clock interrupts per second */)</pre> |
| DESCRIPTION | <p>This routine sets the interrupt rate of the system clock. It is called by <i>usrRoot()</i> in <i>usrConfig.c</i>.</p> <p>There may be interactions between this routine and the POSIX clockLib routines. Refer to the clockLib reference entry.</p> |
| NOTE | <p>This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.</p> |
| RETURNS | OK, or ERROR if the tick rate is invalid or the timer cannot be set. |
| SEE ALSO | <i>sysLib</i> , <i>sysClkEnable()</i> , <i>sysClkRateGet()</i> , clockLib , and BSP-specific reference pages for this routine |

sysHwInit()

| | |
|--------------------|--|
| NAME | <i>sysHwInit()</i> – initialize the system hardware |
| SYNOPSIS | <pre>void sysHwInit (void)</pre> |
| DESCRIPTION | <p>This routine initializes various features of the board. It is called from <i>usrInit()</i> in <i>usrConfig.c</i>.</p> |
| NOTE | <p>This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.</p> |
| NOTE | <p>This routine should not be called directly by the user application.</p> |
| RETURNS | N/A |
| SEE ALSO | <i>sysLib</i> , and BSP-specific reference pages for this routine |

sysIntDisable()

| | |
|--------------------|---|
| NAME | <i>sysIntDisable()</i> – disable a bus interrupt level |
| SYNOPSIS | <pre>STATUS sysIntDisable (int intLevel /* interrupt level to disable */)</pre> |
| DESCRIPTION | This routine disables a specified bus interrupt level. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.* |
| RETURNS | OK, or ERROR if <i>intLevel</i> is out of range. |
| SEE ALSO | <i>sysLib</i> , <i>sysIntEnable()</i> , and BSP-specific reference pages for this routine |

sysIntEnable()

| | |
|--------------------|---|
| NAME | <i>sysIntEnable()</i> – enable a bus interrupt level |
| SYNOPSIS | <pre>STATUS sysIntEnable (int intLevel /* interrupt level to enable (1-7) */)</pre> |
| DESCRIPTION | This routine enables a specified bus interrupt level. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.* |
| RETURNS | OK, or ERROR if <i>intLevel</i> is out of range. |
| SEE ALSO | <i>sysLib</i> , <i>sysIntDisable()</i> , and BSP-specific reference pages for this routine |

sysLocalToBusAdrs()

- NAME** *sysLocalToBusAdrs()* – convert a local address to a bus address
- SYNOPSIS**
- ```
STATUS sysLocalToBusAdrs
(
 int adrsSpace, /* bus address space in which busAdrs resides */
 char * localAdrs, /* local address to convert */
 char * *pBusAdrs /* where to return bus address */
)
```
- DESCRIPTION** This routine gets the bus address that accesses a specified local memory address.
- NOTE** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.
- RETURNS** OK, or ERROR if the address space is unknown or not mapped.
- SEE ALSO** *sysLib*, *sysBusToLocalAdrs()*, and BSP-specific reference pages for this routine

---

## *sysMailboxConnect()*

- NAME** *sysMailboxConnect()* – connect a routine to the mailbox interrupt
- SYNOPSIS**
- ```
STATUS sysMailboxConnect
(
    FUNCPTR routine, /* routine called at each mailbox interrupt */
    int    arg      /* argument with which to call routine */
)
```
- DESCRIPTION** This routine specifies the interrupt service routine to be called at each mailbox interrupt.
- NOTE** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.
- RETURNS** OK, or ERROR if the routine cannot be connected to the interrupt.
- SEE ALSO** *sysLib*, *intConnect()*, *sysMailboxEnable()*, and BSP-specific reference pages for this routine

sysMailboxEnable()

| | |
|--------------------|--|
| NAME | <i>sysMailboxEnable()</i> – enable the mailbox interrupt |
| SYNOPSIS | <pre>STATUS sysMailboxEnable (char * mailboxAdrs /* address of mailbox (ignored) */)</pre> |
| DESCRIPTION | This routine enables the mailbox interrupt. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | OK, always. |
| SEE ALSO | <i>sysLib</i> , <i>sysMailboxConnect()</i> , and BSP-specific reference pages for this routine |

sysMemTop()

| | |
|--------------------|--|
| NAME | <i>sysMemTop()</i> – get the address of the top of logical memory |
| SYNOPSIS | <pre>char *sysMemTop (void)</pre> |
| DESCRIPTION | This routine returns the address of the top of memory. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | The address of the top of memory. |
| SEE ALSO | <i>sysLib</i> , and BSP-specific reference pages for this routine |

sysModel()

| | |
|--------------------|--|
| NAME | <i>sysModel()</i> – return the model name of the CPU board |
| SYNOPSIS | <code>char *sysModel (void)</code> |
| DESCRIPTION | This routine returns the model name of the CPU board. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | A pointer to a string containing the board name. |
| SEE ALSO | <i>sysLib</i> , and BSP-specific reference pages for this routine |

sysNvRamGet()

| | |
|--------------------|--|
| NAME | <i>sysNvRamGet()</i> – get the contents of non-volatile RAM |
| SYNOPSIS | <pre>STATUS sysNvRamGet (char * string, /* where to copy non-volatile RAM */ int strLen, /* maximum number of bytes to copy */ int offset /* byte offset into non-volatile RAM */)</pre> |
| DESCRIPTION | This routine copies the contents of non-volatile memory into a specified string. The string will be terminated with an EOS. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | OK, or ERROR if access is outside the non-volatile RAM address range. |
| SEE ALSO | <i>sysLib</i> , <i>sysNvRamSet()</i> , and BSP-specific reference pages for this routine |

sysNvRamSet()

| | |
|--------------------|---|
| NAME | <i>sysNvRamSet()</i> – write to non-volatile RAM |
| SYNOPSIS | <pre>STATUS sysNvRamSet (char * string, /* string to be copied into non-volatile RAM */ int strLen, /* maximum number of bytes to copy */ int offset /* byte offset into non-volatile RAM */)</pre> |
| DESCRIPTION | This routine copies a specified string into non-volatile RAM. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | OK, or ERROR if access is outside the non-volatile RAM address range. |
| SEE ALSO | <i>sysLib</i> , <i>sysNvRamGet()</i> , and BSP-specific reference pages for this routine |

sysPhysMemTop()

| | |
|--------------------|--|
| NAME | <i>sysPhysMemTop()</i> – get the address of the top of memory |
| SYNOPSIS | <pre>char * sysPhysMemTop (void)</pre> |
| DESCRIPTION | <p>This routine returns the address of the first missing byte of memory, which indicates the top of memory. Normally, the amount of physical memory is specified with the macro <code>LOCAL_MEM_SIZE</code>. BSPs that support run-time memory sizing do so only if the macro <code>LOCAL_MEM_AUTOSIZE</code> is defined. If not defined, then <code>LOCAL_MEM_SIZE</code> is assumed to be, and must be, the true size of physical memory.</p> <p>NOTE: Do not adjust <code>LOCAL_MEM_SIZE</code> to reserve memory for application use. See <i>sysMemTop()</i> for more information on reserving memory.</p> |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | The address of the top of physical memory. |

SEE ALSO *sysLib*, *sysMemTop()*, and BSP-specific reference pages for this routine

sysProcNumGet()

NAME *sysProcNumGet()* – get the processor number

SYNOPSIS `int sysProcNumGet (void)`

DESCRIPTION This routine returns the processor number for the CPU board, which is set with *sysProcNumSet()*.

NOTE This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

RETURNS The processor number for the CPU board.

SEE ALSO *sysLib*, *sysProcNumSet()*, and BSP-specific reference pages for this routine

sysProcNumSet()

NAME *sysProcNumSet()* – set the processor number

SYNOPSIS `void sysProcNumSet
(
 int procNum /* processor number */
)`

DESCRIPTION This routine sets the processor number for the CPU board. Processor numbers should be unique on a single backplane.

NOTE This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

RETURNS N/A

SEE ALSO *sysLib*, *sysProcNumGet()*, and BSP-specific reference pages for this routine

sysScsiBusReset()

| | |
|--------------------|---|
| NAME | <i>sysScsiBusReset()</i> – assert the RST line on the SCSI bus (Western Digital WD33C93 only) |
| SYNOPSIS | <pre>void sysScsiBusReset (WD_33C93_SCSI_CTRL * pSbic /* ptr to SBIC info */)</pre> |
| DESCRIPTION | This routine asserts the RST line on the SCSI bus, which causes all connected devices to return to a quiescent state. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | N/A |
| SEE ALSO | <i>sysLib</i> , and BSP-specific reference pages for this routine |

sysScsiConfig()

| | |
|--------------------|--|
| NAME | <i>sysScsiConfig()</i> – system SCSI configuration |
| SYNOPSIS | <code>STATUS sysScsiConfig (void)</code> |
| DESCRIPTION | <p>This is an example SCSI configuration routine.</p> <p>Most of the code found here is an example of how to declare a SCSI peripheral configuration. You must edit this routine to reflect the actual configuration of your SCSI bus. This example can also be found in <code>src/config/usrScsi.c</code>.</p> <p>If you are just getting started, you can test your hardware configuration by defining <code>SCSI_AUTO_CONFIG</code>, which will probe the bus and display all devices found. No device should have the same SCSI bus ID as your VxWorks SCSI port (default = 7), or the same as any other device. Check for proper bus termination.</p> <p>There are three configuration examples here. They demonstrate configuration of a SCSI hard disk (any type), an OMTI 3500 floppy disk, and a tape drive (any type).</p> |
| Hard Disk | The hard disk is divided into two 32-Mbyte partitions and a third partition with the remainder of the disk. The first partition is initialized as a dosFs device. The second and third partitions are initialized as rt11Fs devices, each with 256 directory entries. |

It is recommended that the first partition (**BLK_DEV**) on a block device be a dosFs device, if the intention is eventually to boot VxWorks from the device. This will simplify the task considerably.

Floppy Disk

The floppy, since it is a removable medium device, is allowed to have only a single partition, and dosFs is the file system of choice for this device, since it facilitates media compatibility with IBM PC machines.

In contrast to the hard disk configuration, the floppy setup in this example is more intricate. Note that the *scsiPhysDevCreate()* call is issued twice. The first time is merely to get a "handle" to pass to *scsiModeSelect()*, since the default media type is sometimes inappropriate (in the case of generic SCSI-to-floppy cards). After the hardware is correctly configured, the handle is discarded via *scsiPhysDevDelete()*, after which the peripheral is correctly configured by a second call to *scsiPhysDevCreate()*. (Before the *scsiModeSelect()* call, the configuration information was incorrect.) Note that after the *scsiBlkDevCreate()* call, the correct values for *sectorsPerTrack* and *nHeads* must be set via *scsiBlkDevInit()*. This is necessary for IBM PC compatibility.

Tape Drive

The tape configuration is also somewhat complex because certain device parameters need to be turned off within VxWorks and the fixed-block size needs to be defined, assuming that the tape supports fixed blocks.

The last parameter to the *dosFsDevInit()* call is a pointer to a **DOS_VOL_CONFIG** structure. By specifying **NULL**, you are asking *dosFsDevInit()* to read this information off the disk in the drive. This may fail if no disk is present or if the disk has no valid dosFs directory. Should this be the case, you can use the *dosFsMkfs()* command to create a new directory on a disk. This routine uses default parameters (see **dosFsLib**) that may not be suitable for your application, in which case you should use *dosFsDevInit()* with a pointer to a valid **DOS_VOL_CONFIG** structure that you have created and initialized. If *dosFsDevInit()* is used, a *diskInit()* call should be made to write a new directory on the disk, if the disk is blank or disposable.

NOTE

The variable **pSbdFloppy** is global to allow the above calls to be made from the VxWorks shell, for example:

```
-> dosFsMkfs "/fd0/", pSbdFloppy
```

If a disk is new, use *diskFormat()* to format it.

NOTE

This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

RETURNS

OK or ERROR.

SEE ALSO

sysLib, and BSP-specific reference pages for this routine

sysScsiInit()

| | |
|--------------------|--|
| NAME | <i>sysScsiInit()</i> – initialize an on-board SCSI port |
| SYNOPSIS | <code>STATUS sysScsiInit (void)</code> |
| DESCRIPTION | <p>This routine creates and initializes a SCSI control structure, enabling use of the on-board SCSI port. It also connects the proper interrupt service routine to the desired vector, and enables the interrupt at the desired level.</p> <p>If SCSI DMA is supported by the board and <code>INCLUDE_SCSI_DMA</code> is defined, the DMA is also initialized.</p> |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | OK, or ERROR if the control structure cannot be connected, the controller cannot be initialized, or the DMA's interrupt cannot be connected. |
| SEE ALSO | <code>sysLib</code> , and BSP-specific reference pages for this routine |

sysSerialChanGet()

| | |
|--------------------|---|
| NAME | <i>sysSerialChanGet()</i> – get the SIO_CHAN device associated with a serial channel |
| SYNOPSIS | <pre>SIO_CHAN * sysSerialChanGet (int channel /* serial channel */)</pre> |
| DESCRIPTION | This routine gets the SIO_CHAN device associated with a specified serial channel. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | A pointer to the SIO_CHAN structure for the channel, or ERROR if the channel is invalid. |
| SEE ALSO | <code>sysLib</code> , and BSP-specific reference pages for this routine |

sysSerialHwInit()

| | |
|--------------------|---|
| NAME | <i>sysSerialHwInit()</i> – initialize the BSP serial devices to a quiescent state |
| SYNOPSIS | <code>void sysSerialHwInit (void)</code> |
| DESCRIPTION | This routine initializes the BSP serial device descriptors and puts the devices in a quiescent state. It is called from <i>sysHwInit()</i> with interrupts locked. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | N/A |
| SEE ALSO | <i>sysLib</i> , and BSP-specific reference pages for this routine |

sysSerialHwInit2()

| | |
|--------------------|--|
| NAME | <i>sysSerialHwInit2()</i> – connect BSP serial device interrupts |
| SYNOPSIS | <code>void sysSerialHwInit2 (void)</code> |
| DESCRIPTION | This routine connects the BSP serial device interrupts. It is called from <i>sysHwInit2()</i> . Serial device interrupts could not be connected in <i>sysSerialHwInit()</i> because the kernel memory allocator was not initialized at that point, and <i>intConnect()</i> calls <i>malloc()</i> . |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | N/A |
| SEE ALSO | <i>sysLib</i> , and BSP-specific reference pages for this routine |

sysSerialReset()

| | |
|--------------------|--|
| NAME | <i>sysSerialReset()</i> – reset all SIO devices to a quiet state |
| SYNOPSIS | <code>void sysSerialReset (void)</code> |
| DESCRIPTION | This routine is called from <i>sysToMonitor()</i> to reset all SIO device and prevent them from generating interrupts or performing DMA cycles. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | N/A |
| SEE ALSO | <code>sysLib</code> , and BSP-specific reference pages for this routine |

system()

| | |
|----------------------|---|
| NAME | <i>system()</i> – pass a string to a command processor (Unimplemented) (ANSI) |
| SYNOPSIS | <pre>int system (const char * string /* pointer to string */)</pre> |
| DESCRIPTION | This function is not applicable to VxWorks. |
| INCLUDE FILES | <code>stdlib.h</code> |
| RETURNS | OK, always. |
| SEE ALSO | <code>ansiStdlib</code> |

sysToMonitor()

| | |
|--------------------|---|
| NAME | <i>sysToMonitor()</i> – transfer control to the ROM monitor |
| SYNOPSIS | <pre>STATUS sysToMonitor (int startType /* parameter passed to ROM to tell it how to boot */)</pre> |
| DESCRIPTION | This routine transfers control to the ROM monitor. Normally, it is called only by <i>reboot()</i> --which services CTRL+X--and by bus errors at interrupt level. However, in some circumstances, the user may wish to introduce a <i>startType</i> to enable special boot ROM facilities. |
| NOTE | This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this routine is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP. |
| RETURNS | Does not return. |
| SEE ALSO | <i>sysLib</i> , and BSP-specific reference pages for this routine |

tan()

| | |
|----------------------|--|
| NAME | <i>tan()</i> – compute a tangent (ANSI) |
| SYNOPSIS | <pre>double tan (double x /* angle in radians */)</pre> |
| DESCRIPTION | This routine computes the tangent of <i>x</i> in double precision. The angle <i>x</i> is expressed in radians. |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision tangent of <i>x</i> . |
| SEE ALSO | ansiMath, mathALib |

tanf()

| | |
|----------------------|---|
| NAME | <i>tanf()</i> – compute a tangent (ANSI) |
| SYNOPSIS | <pre>float tanf (float x /* angle in radians */)</pre> |
| DESCRIPTION | This routine returns the tangent of x in single precision. The angle x is expressed in radians. |
| INCLUDE FILES | math.h |
| RETURNS | The single-precision tangent of x . |
| SEE ALSO | mathALib |

tanh()

| | |
|----------------------|---|
| NAME | <i>tanh()</i> – compute a hyperbolic tangent (ANSI) |
| SYNOPSIS | <pre>double tanh (double x /* number whose hyperbolic tangent is required */)</pre> |
| DESCRIPTION | This routine returns the hyperbolic tangent of x in double precision (IEEE double, 53 bits). |
| INCLUDE FILES | math.h |
| RETURNS | The double-precision hyperbolic tangent of x . Special cases: If x is NaN, <i>tanh()</i> returns NaN. |
| SEE ALSO | ansiMath, mathALib |

tanhf()

tanhf()

| | |
|----------------------|--|
| NAME | <i>tanhf()</i> – compute a hyperbolic tangent (ANSI) |
| SYNOPSIS | <pre>float tanhf (float x /* number whose hyperbolic tangent is required */)</pre> |
| DESCRIPTION | This routine returns the hyperbolic tangent of <i>x</i> in single precision. |
| INCLUDE FILES | math.h |
| RETURNS | The single-precision hyperbolic tangent of <i>x</i> . |
| SEE ALSO | mathALib |

tapeFsDevInit()

| | |
|--------------------|--|
| NAME | <i>tapeFsDevInit()</i> – associate a sequential device with tape volume functions |
| SYNOPSIS | <pre>TAPE_VOL_DESC *tapeFsDevInit (char * volName, /* volume name */ SEQ_DEV * pSeqDev, /* pointer to sequential device info */ TAPE_CONFIG * pTapeConfig /* pointer to tape config info */)</pre> |
| DESCRIPTION | <p>This routine takes a sequential device created by a device driver and defines it as a tape file system volume. As a result, when high-level I/O operations, such as <i>open()</i> and <i>write()</i>, are performed on the device, the calls will be routed through tapeFsLib.</p> <p>This routine associates volName with a device and installs it in the VxWorks I/O system-device table. The driver number used when the device is added to the table is that which was assigned to the tape library during <i>tapeFsInit()</i>. (The driver number is kept in the global variable tapeFsDrvNum.)</p> <p>The SEQ_DEV structure specified by pSeqDev contains configuration data describing the device and the addresses of the routines which are called to read blocks, write blocks, write file marks, reset the device, check device status, perform other I/O control functions (<i>ioctl()</i>), reserve and release devices, load and unload devices, and rewind devices. These</p> |

routines are not called until they are required by subsequent I/O operations. The TAPE_CONFIG structure is used to define configuration parameters for the TAPE_VOL_DESC. The configuration parameters are defined and described in **tapeFsLib.h**.

- RETURNS** A pointer to the volume descriptor (TAPE_VOL_DESC), or NULL if there is an error.
- ERRNO** S_tapeFsLib_NO_SEQ_DEV, S_tapeFsLib_ILLEGAL_TAPE_CONFIG_PARM
- SEE ALSO** **tapeFsLib**

tapeFsInit()

- NAME** *tapeFsInit()* – initialize the tape volume library
- SYNOPSIS** `STATUS tapeFsInit ()`
- DESCRIPTION** This routine initializes the tape volume library. It must be called exactly once, before any other routine in the library. Only one file descriptor per volume is assumed.
- This routine also installs tape volume library routines in the VxWorks I/O system driver table. The driver number assigned to **tapeFsLib** is placed in the global variable **tapeFsDrvNum**. This number is later associated with system file descriptors opened to tapeFs devices.
- To enable this initialization, simply call the routine *tapeFsDevInit()*, which automatically calls *tapeFsInit()* in order to initialize the tape file system.
- RETURNS** OK or ERROR.
- SEE ALSO** **tapeFsLib**

tapeFsReadyChange()

- NAME** *tapeFsReadyChange()* – notify **tapeFsLib** of a change in ready status
- SYNOPSIS** `STATUS tapeFsReadyChange
(
 TAPE_VOL_DESC * pTapeVol /* pointer to volume descriptor */
)`

| | |
|--------------------|--|
| DESCRIPTION | <p>This routine sets the volume descriptor state to TAPE_VD_READY_CHANGED. It should be called whenever a driver senses that a device has come on-line or gone off-line (for example, that a tape has been inserted or removed).</p> <p>After this routine has been called, the next attempt to use the volume results in an attempted remount.</p> |
| RETURNS | OK if the read change status is set, or ERROR if the file descriptor is in use. |
| ERRNO | S_tapeFsLib_FILE_DESCRIPTOR_BUSY |
| SEE ALSO | tapeFsLib |

tapeFsVolUnmount()

| | |
|--------------------|---|
| NAME | <i>tapeFsVolUnmount()</i> – disable a tape device volume |
| SYNOPSIS | <pre>STATUS tapeFsVolUnmount (TAPE_VOL_DESC * pTapeVol /* pointer to volume descriptor */)</pre> |
| DESCRIPTION | <p>This routine is called when I/O operations on a volume are to be discontinued. This is commonly done before changing removable tape. All buffered data for the volume is written to the device (if possible), any open file descriptors are marked obsolete, and the volume is marked not mounted.</p> <p>Because this routine flushes data from memory to the physical device, it should not be used in situations where the tape-change is not recognized until after a new tape has been inserted. In these circumstances, use the ready-change mechanism. (See the manual entry for <i>tapeFsReadyChange()</i>.)</p> <p>This routine may also be called by issuing an <i>ioctl()</i> call using the FIOUNMOUNT function code.</p> |
| RETURNS | OK, or ERROR if the routine cannot access the volume. |
| ERRNO | S_tapeFsLib_VOLUME_NOT_AVAILABLE , S_tapeFsLib_FILE_DESCRIPTOR_BUSY , S_tapeFsLib_SERVICE_NOT_AVAILABLE |
| SEE ALSO | tapeFsLib , <i>tapeFsReadyChange()</i> |

taskActivate()

| | |
|--------------------|---|
| NAME | <i>taskActivate()</i> – activate a task that has been initialized |
| SYNOPSIS | <pre>STATUS taskActivate (int tid /* task ID of task to activate */)</pre> |
| DESCRIPTION | <p>This routine activates tasks created by <i>taskInit()</i>. Without activation, a task is ineligible for CPU allocation by the scheduler. The <i>tid</i> (task ID) argument is simply the address of the WIND_TCB for the task (the <i>taskInit()</i> <i>pTcb</i> argument), cast to an integer:</p> <pre>tid = (int) pTcb;</pre> <p>The <i>taskSpawn()</i> routine is built from <i>taskActivate()</i> and <i>taskInit()</i>. Tasks created by <i>taskSpawn()</i> do not require explicit task activation.</p> |
| RETURNS | OK, or ERROR if the task cannot be activated. |
| SEE ALSO | <i>taskLib</i> , <i>taskInit()</i> |

taskCreateHookAdd()

| | |
|--------------------|---|
| NAME | <i>taskCreateHookAdd()</i> – add a routine to be called at every task create |
| SYNOPSIS | <pre>STATUS taskCreateHookAdd (FUNCPTR createHook /* routine to be called when a task is created */)</pre> |
| DESCRIPTION | <p>This routine adds a specified routine to a list of routines that will be called whenever a task is created. The routine should be declared as follows:</p> <pre>void createHook (WIND_TCB *pNewTcb /* pointer to new task's TCB */)</pre> |
| RETURNS | OK, or ERROR if the table of task create routines is full. |
| SEE ALSO | <i>taskHookLib</i> , <i>taskCreateHookDelete()</i> |

taskCreateHookDelete()

NAME *taskCreateHookDelete()* – delete a previously added task create routine

SYNOPSIS `STATUS taskCreateHookDelete`
`(`
`FUNCPTR createHook /* routine to be deleted from list */`
`)`

DESCRIPTION This routine removes a specified routine from the list of routines to be called at each task create.

RETURNS OK, or ERROR if the routine is not in the table of task create routines.

SEE ALSO *taskHookLib*, *taskCreateHookAdd()*

taskCreateHookShow()

NAME *taskCreateHookShow()* – show the list of task create routines

SYNOPSIS `void taskCreateHookShow (void)`

DESCRIPTION This routine shows all the task create routines installed in the task create hook table, in the order in which they were installed.

RETURNS N/A

SEE ALSO *taskHookShow*, *taskCreateHookAdd()*

taskDelay()

NAME *taskDelay()* – delay a task from executing

SYNOPSIS `STATUS taskDelay`
`(`
`int ticks /* number of ticks to delay task */`
`)`

| | |
|--------------------|---|
| DESCRIPTION | This routine causes the calling task to relinquish the CPU for the duration specified (in ticks). This is commonly referred to as manual rescheduling, but it is also useful when waiting for some external condition that does not have an interrupt associated with it. If the calling task receives a signal that is not being blocked or ignored, <i>taskDelay()</i> returns ERROR and sets errno to EINTR after the signal handler is run. |
| RETURNS | OK, or ERROR if called from interrupt level or if the calling task receives a signal that is not blocked or ignored. |
| ERRNO | S_intLib_NOT_ISR_CALLABLE, EINTR |
| SEE ALSO | taskLib |

taskDelete()

| | |
|--------------------|---|
| NAME | <i>taskDelete()</i> – delete a task |
| SYNOPSIS | <pre> STATUS taskDelete (int tid /* task ID of task to delete */) </pre> |
| DESCRIPTION | This routine causes a specified task to cease to exist and deallocates the stack and WIND_TCB memory resources. Upon deletion, all routines specified by <i>taskDeleteHookAdd()</i> will be called in the context of the deleting task. This routine is the companion routine to <i>taskSpawn()</i> . |
| RETURNS | OK, or ERROR if the task cannot be deleted. |
| ERRNO | S_intLib_NOT_ISR_CALLABLE, S_objLib_OBJ_DELETED S_objLib_OBJ_UNAVAILABLE, S_objLib_OBJ_ID_ERROR |
| SEE ALSO | taskLib, excLib, <i>taskDeleteHookAdd()</i> , <i>taskSpawn()</i> , <i>VxWorks Programmer's Guide: Basic OS</i> |

taskDeleteForce()

| | |
|--------------------|--|
| NAME | <i>taskDeleteForce()</i> – delete a task without restriction |
| SYNOPSIS | <pre>STATUS taskDeleteForce (int tid /* task ID of task to delete */)</pre> |
| DESCRIPTION | This routine deletes a task even if the task is protected from deletion. It is similar to <i>taskDelete()</i> . Upon deletion, all routines specified by <i>taskDeleteHookAdd()</i> will be called in the context of the deleting task. |
| CAVEATS | <p>This routine is intended as a debugging aid, and is generally inappropriate for applications. Disregarding a task's deletion protection could leave the the system in an unstable state or lead to system deadlock.</p> <p>The system does not protect against simultaneous <i>taskDeleteForce()</i> calls. Such a situation could leave the system in an unstable state.</p> |
| RETURNS | OK, or ERROR if the task cannot be deleted. |
| ERRNO | S_intLib_NOT_ISR_CALLABLE, S_objLib_OBJ_DELETED, S_objLib_OBJ_UNAVAILABLE, S_objLib_OBJ_ID_ERROR |
| SEE ALSO | <i>taskLib</i> , <i>taskDeleteHookAdd()</i> , <i>taskDelete()</i> |

taskDeleteHookAdd()

| | |
|--------------------|--|
| NAME | <i>taskDeleteHookAdd()</i> – add a routine to be called at every task delete |
| SYNOPSIS | <pre>STATUS taskDeleteHookAdd (FUNCPTR deleteHook /* routine to be called when a task is deleted */)</pre> |
| DESCRIPTION | <p>This routine adds a specified routine to a list of routines that will be called whenever a task is deleted. The routine should be declared as follows:</p> <pre>void deleteHook (</pre> |


```
WIND_TCB *pTcb      /* pointer to deleted task's WIND_TCB */  
)
```

RETURNS OK, or ERROR if the table of task delete routines is full.

SEE ALSO *taskHookLib*, *taskDeleteHookDelete()*

taskDeleteHookDelete()

NAME *taskDeleteHookDelete()* – delete a previously added task delete routine

SYNOPSIS

```
STATUS taskDeleteHookDelete  
(  
    FUNCPTR deleteHook /* routine to be deleted from list */  
)
```

DESCRIPTION This routine removes a specified routine from the list of routines to be called at each task delete.

RETURNS OK, or ERROR if the routine is not in the table of task delete routines.

SEE ALSO *taskHookLib*, *taskDeleteHookAdd()*

taskDeleteHookShow()

NAME *taskDeleteHookShow()* – show the list of task delete routines

SYNOPSIS

```
void taskDeleteHookShow (void)
```

DESCRIPTION This routine shows all the delete routines installed in the task delete hook table, in the order in which they were installed. Note that the delete routines will be run in reverse of the order in which they were installed.

RETURNS N/A

SEE ALSO *taskHookShow*, *taskDeleteHookAdd()*

taskHookInit()

| | |
|--------------------|---|
| NAME | <i>taskHookInit()</i> – initialize task hook facilities |
| SYNOPSIS | <code>void taskHookInit (void)</code> |
| DESCRIPTION | This routine is a NULL routine called to configure the task hook package into the system. It is called automatically if the configuration macro INCLUDE_TASK_HOOKS is defined. |
| RETURNS | N/A |
| SEE ALSO | taskHookLib |

taskHookShowInit()

| | |
|--------------------|--|
| NAME | <i>taskHookShowInit()</i> – initialize the task hook show facility |
| SYNOPSIS | <code>void taskHookShowInit (void)</code> |
| DESCRIPTION | <p>This routine links the task hook show facility into the VxWorks system. It is called automatically when the task hook show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none">– If you use the configuration header files, define INCLUDE_SHOW_ROUTINES in config.h.– If you use the Tornado project facility, select INCLUDE_TASK_HOOK_SHOW. |
| RETURNS | N/A |
| SEE ALSO | taskHookShow |

taskIdDefault()

| | |
|--------------------|--|
| NAME | <i>taskIdDefault()</i> – set the default task ID |
| SYNOPSIS | <pre>int taskIdDefault (int tid /* user supplied task ID; if 0, return default */)</pre> |
| DESCRIPTION | <p>This routine maintains a global default task ID. This ID is used by libraries that want to allow a task ID argument to take on a default value when one is not explicitly supplied.</p> <p>If <i>tid</i> is not zero (i.e., the user did specify a task ID), the default ID is set to that value, and that value is returned. If <i>tid</i> is zero (i.e., the user did not specify a task ID), the default ID is not changed and its value is returned. Thus the value returned is always the last task ID the user specified.</p> |
| RETURNS | The most recent non-zero task ID. |
| SEE ALSO | taskInfo , dbgLib , <i>VxWorks Programmer's Guide: Target Shell</i> , windsh , <i>Tornado User's Guide: Shell</i> |

taskIdListGet()

| | |
|--------------------|---|
| NAME | <i>taskIdListGet()</i> – get a list of active task IDs |
| SYNOPSIS | <pre>int taskIdListGet (int idList[], /* array of task IDs to be filled in */ int maxTasks /* max tasks idList can accommodate */)</pre> |
| DESCRIPTION | <p>This routine provides the calling task with a list of all active tasks. An unsorted list of task IDs for no more than <i>maxTasks</i> tasks is put into <i>idList</i>.</p> |
| WARNING | <p>Kernel rescheduling is disabled with taskLock() while tasks are filled into the <i>idList</i>. There is no guarantee that all the tasks are valid or that new tasks have not been created by the time this routine returns.</p> |
| RETURNS | The number of tasks put into the ID list. |
| SEE ALSO | taskInfo |

taskIdSelf()

| | |
|--------------------|--|
| NAME | <i>taskIdSelf()</i> – get the task ID of a running task |
| SYNOPSIS | <code>int taskIdSelf (void)</code> |
| DESCRIPTION | This routine gets the task ID of the calling task. The task ID will be invalid if called at interrupt level. |
| RETURNS | The task ID of the calling task. |
| SEE ALSO | <code>taskLib</code> |

taskIdVerify()

| | |
|--------------------|--|
| NAME | <i>taskIdVerify()</i> – verify the existence of a task |
| SYNOPSIS | <code>STATUS taskIdVerify (int tid /* task ID */)</code> |
| DESCRIPTION | This routine verifies the existence of a specified task by validating the specified ID as a task ID. |
| RETURNS | OK, or ERROR if the task ID is invalid. |
| ERRNO | <code>S_objLib_OBJ_ID_ERROR</code> |
| SEE ALSO | <code>taskLib</code> |

taskInfoGet()

| | |
|--------------------|--|
| NAME | <i>taskInfoGet()</i> – get information about a task |
| SYNOPSIS | <pre>STATUS taskInfoGet (int tid, /* ID of task for which to get info */ TASK_DESC * pTaskDesc /* task descriptor to be filled in */)</pre> |
| DESCRIPTION | This routine fills in a specified task descriptor (<code>TASK_DESC</code>) for a specified task. The information in the task descriptor is, for the most part, a copy of information kept in the task control block (<code>WIND_TCB</code>). The <code>TASK_DESC</code> structure is useful for common information and avoids dealing directly with the unwieldy <code>WIND_TCB</code> . |
| NOTE | Examination of <code>WIND_TCBs</code> should be restricted to debugging aids. |
| RETURNS | OK, or ERROR if the task ID is invalid. |
| SEE ALSO | <i>taskShow</i> |

taskInit()

| | |
|-----------------|--|
| NAME | <i>taskInit()</i> – initialize a task with a stack at a specified address |
| SYNOPSIS | <pre>STATUS taskInit (WIND_TCB * pTcb, /* address of new task's TCB */ char * name, /* name of new task (stored at pStackBase) */ int priority, /* priority of new task */ int options, /* task option word */ char * pStackBase, /* base of new task's stack */ int stackSize, /* size (bytes) of stack needed */ FUNCPTR entryPt, /* entry point of new task */ int arg1, /* first of ten task args to pass to func */ int arg2, int arg3, int arg4, int arg5, int arg6, int arg7,</pre> |

taskIsReady()

```

    int      arg8,
    int      arg9,
    int      arg10
)

```

DESCRIPTION This routine initializes user-specified regions of memory for a task stack and control block instead of allocating them from memory as *taskSpawn()* does. This routine will utilize the specified pointers to the `WIND_TCB` and stack as the components of the task. This allows, for example, the initialization of a static `WIND_TCB` variable. It also allows for special stack positioning as a debugging aid.

As in *taskSpawn()*, a task may be given a name. While *taskSpawn()* automatically names unnamed tasks, *taskInit()* permits the existence of tasks without names. The task ID required by other task routines is simply the address *pTcb*, cast to an integer.

Note that the task stack may grow up or down from `pStackBase`, depending on the target architecture.

Other arguments are the same as in *taskSpawn()*. Unlike *taskSpawn()*, *taskInit()* does not activate the task. This must be done by calling *taskActivate()* after calling *taskInit()*.

Normally, tasks should be started using *taskSpawn()* rather than *taskInit()*, except when additional control is required for task memory allocation or a separate task activation is desired.

RETURNS OK, or ERROR if the task cannot be initialized.

ERRNO `S_intLib_NOT_ISR_CALLABLE`, `S_objLib_OBJ_ID_ERROR`

SEE ALSO *taskLib*, *taskActivate()*, *taskSpawn()*

taskIsReady()

NAME *taskIsReady()* – check if a task is ready to run

SYNOPSIS

```

BOOL taskIsReady
(
    int tid /* task ID */
)

```

DESCRIPTION This routine tests the status field of a task to determine if it is ready to run.

RETURNS TRUE if the task is ready, otherwise FALSE.

SEE ALSO *taskInfo*

taskIsSuspended()

NAME *taskIsSuspended()* – check if a task is suspended

SYNOPSIS

```
BOOL taskIsSuspended
(
    int tid /* task ID */
)
```

DESCRIPTION This routine tests the status field of a task to determine if it is suspended.

RETURNS TRUE if the task is suspended, otherwise FALSE.

SEE ALSO *taskInfo*

taskLock()

NAME *taskLock()* – disable task rescheduling

SYNOPSIS `STATUS taskLock (void)`

DESCRIPTION This routine disables task context switching. The task that calls this routine will be the only task that is allowed to execute, unless the task explicitly gives up the CPU by making itself no longer ready. Typically this call is paired with *taskUnlock()*; together they surround a critical section of code. These preemption locks are implemented with a counting variable that allows nested preemption locks. Preemption will not be unlocked until *taskUnlock()* has been called as many times as *taskLock()*.

This routine does not lock out interrupts; use *intLock()* to lock out interrupts.

A *taskLock()* is preferable to *intLock()* as a means of mutual exclusion, because interrupt lock-outs add interrupt latency to the system.

A *semTake()* is preferable to *taskLock()* as a means of mutual exclusion, because preemption lock-outs add preemptive latency to the system.

The *taskLock()* routine is not callable from interrupt service routines.

RETURNS OK or ERROR.

ERRNO `S_objLib_OBJ_ID_ERROR`, `S_intLib_NOT_ISR_CALLABLE`

SEE ALSO *taskLib*, *taskUnlock()*, *intLock()*, *taskSafe()*, *semTake()*

taskName()

taskName()

| | |
|--------------------|--|
| NAME | <i>taskName()</i> – get the name associated with a task ID |
| SYNOPSIS | <pre>char *taskName (int tid /* ID of task whose name is to be found */)</pre> |
| DESCRIPTION | This routine returns a pointer to the name of a task of a specified ID, if the task has a name. If the task has no name, it returns an empty string. |
| RETURNS | A pointer to the task name, or NULL if the task ID is invalid. |
| SEE ALSO | taskInfo |

taskNameToId()

| | |
|--------------------|--|
| NAME | <i>taskNameToId()</i> – look up the task ID associated with a task name |
| SYNOPSIS | <pre>int taskNameToId (char * name /* task name to look up */)</pre> |
| DESCRIPTION | This routine returns the ID of the task matching a specified name. Referencing a task in this way is inefficient, since it involves a search of the task list. |
| RETURNS | The task ID, or ERROR if the task is not found. |
| ERRNO | S_taskLib_NAME_NOT_FOUND |
| SEE ALSO | taskInfo |

taskOptionsGet()

NAME *taskOptionsGet()* – examine task options

SYNOPSIS

```
STATUS taskOptionsGet
(
    int    tid,        /* task ID */
    int *  pOptions /* task's options */
)
```

DESCRIPTION This routine gets the current execution options of the specified task. The option bits returned by this routine indicate the following modes:

VX_FP_TASK
execute with floating-point coprocessor support.

VX_PRIVATE_ENV
include private environment support (see **envLib**).

VX_NO_STACK_FILL
do not fill the stack for use by *checkstack()*.

VX_UNBREAKABLE
do not allow breakpoint debugging.

For definitions, see **taskLib.h**.

RETURNS OK, or ERROR if the task ID is invalid.

SEE ALSO *taskInfo*, *taskOptionsSet()*

taskOptionsSet()

NAME *taskOptionsSet()* – change task options

SYNOPSIS

```
STATUS taskOptionsSet
(
    int tid,        /* task ID */
    int mask,       /* bit mask of option bits to unset */
    int newOptions /* bit mask of option bits to set */
)
```

DESCRIPTION This routine changes the execution options of a task. The only option that can be changed after a task has been created is:

VX_UNBREAKABLE
do not allow breakpoint debugging.

For definitions, see **taskLib.h**.

RETURNS OK, or ERROR if the task ID is invalid.

SEE ALSO **taskInfo**, **taskOptionsGet()**

taskPriorityGet()

NAME ***taskPriorityGet()*** – examine the priority of a task

SYNOPSIS

```
STATUS taskPriorityGet
(
    int  tid,          /* task ID */
    int * pPriority /* return priority here */
)
```

DESCRIPTION This routine determines the current priority of a specified task. The current priority is copied to the integer pointed to by *pPriority*.

RETURNS OK, or ERROR if the task ID is invalid.

ERRNO S_objLib_OBJ_ID_ERROR

SEE ALSO **taskLib**, **taskPrioritySet()**

taskPrioritySet()

NAME ***taskPrioritySet()*** – change the priority of a task

SYNOPSIS

```
STATUS taskPrioritySet
(
    int tid,          /* task ID */
    int newPriority /* new priority */
)
```

| | |
|--------------------|---|
| DESCRIPTION | This routine changes a task's priority to a specified priority. Priorities range from 0, the highest priority, to 255, the lowest priority. |
| RETURNS | OK, or ERROR if the task ID is invalid. |
| ERRNO | S_taskLib_ILLEGAL_PRIORITY, S_objLib_OBJ_ID_ERROR |
| SEE ALSO | taskLib, taskPriorityGet() |

taskRegsGet()

| | |
|--------------------|---|
| NAME | <i>taskRegsGet()</i> – get a task's registers from the TCB |
| SYNOPSIS | <pre>STATUS taskRegsGet (int tid, /* task ID */ REG_SET * pRegs /* put register contents here */)</pre> |
| DESCRIPTION | This routine gathers task information kept in the TCB. It copies the contents of the task's registers to the register structure <i>pRegs</i> . |
| NOTE | This routine only works well if the task is known to be in a stable, non-executing state. Self-examination, for instance, is not advisable, as results are unpredictable. |
| RETURNS | OK, or ERROR if the task ID is invalid. |
| SEE ALSO | taskInfo, taskSuspend(), taskRegsSet() |

taskRegsSet()

| | |
|-----------------|---|
| NAME | <i>taskRegsSet()</i> – set a task's registers |
| SYNOPSIS | <pre>STATUS taskRegsSet (int tid, /* task ID */ REG_SET * pRegs /* get register contents from here */)</pre> |

- DESCRIPTION** This routine loads a specified register set *pRegs* into a specified task's TCB.
- NOTE** This routine only works well if the task is known not to be in the ready state. Suspending the task before changing the register set is recommended.
- RETURNS** OK, or ERROR if the task ID is invalid.
- SEE ALSO** *taskInfo*, *taskSuspend()*, *taskRegsGet()*

taskRegsShow()

NAME *taskRegsShow()* – display the contents of a task's registers

SYNOPSIS

```
void taskRegsShow
(
    int tid /* task ID */
)
```

DESCRIPTION This routine displays the register contents of a specified task on standard output.

EXAMPLE The following example displays the register of the shell task (68000 family):

```
-> taskRegsShow (taskNameToId ("tShell"))
d0   =      0   d1   =      0   d2   =   578fe   d3   =      1
d4   =   3e84e1  d5   =   3e8568  d6   =      0   d7   =   ffffffff
a0   =      0   a1   =      0   a2   =   4f06c   a3   =   578d0
a4   =   3fffc4  a5   =      0   fp   =   3e844c   sp   =   3e842c
sr   =      3000  pc   =   4f0f2
```

RETURNS N/A

SEE ALSO *taskShow*

taskRestart()

| | |
|--------------------|---|
| NAME | <i>taskRestart()</i> – restart a task |
| SYNOPSIS | <pre>STATUS taskRestart (int tid /* task ID of task to restart */)</pre> |
| DESCRIPTION | This routine "restarts" a task. The task is first terminated, and then reinitialized with the same ID, priority, options, original entry point, stack size, and parameters it had when it was terminated. Self-restarting of a calling task is performed by the exception task. The shell utilizes this routine to restart itself when aborted. |
| NOTE | If the task has modified any of its start-up parameters, the restarted task will start with the changed values. |
| RETURNS | OK, or ERROR if the task ID is invalid or the task could not be restarted. |
| ERRNO | S_intLib_NOT_ISR_CALLABLE, S_objLib_OBJ_DELETED, S_objLib_OBJ_UNAVAILABLE, S_objLib_OBJ_ID_ERROR, S_smObjLib_NOT_INITIALIZED, S_memLib_NOT_ENOUGH_MEMORY, S_memLib_BLOCK_ERROR |
| SEE ALSO | taskLib |

taskResume()

| | |
|--------------------|---|
| NAME | <i>taskResume()</i> – resume a task |
| SYNOPSIS | <pre>STATUS taskResume (int tid /* task ID of task to resume */)</pre> |
| DESCRIPTION | This routine resumes a specified task. Suspension is cleared, and the task operates in the remaining state. |
| RETURNS | OK, or ERROR if the task cannot be resumed. |
| ERRNO | S_objLib_OBJ_ID_ERROR |
| SEE ALSO | taskLib |

taskSafe()

NAME *taskSafe()* – make the calling task safe from deletion

SYNOPSIS `STATUS taskSafe (void)`

DESCRIPTION This routine protects the calling task from deletion. Tasks that attempt to delete a protected task will block until the task is made unsafe, using *taskUnsafe()*. When a task becomes unsafe, the deleter will be unblocked and allowed to delete the task.

The *taskSafe()* primitive utilizes a count to keep track of nested calls for task protection. When nesting occurs, the task becomes unsafe only after the outermost *taskUnsafe()* is executed.

RETURNS OK.

SEE ALSO *taskLib*, *taskUnsafe()*, *VxWorks Programmer's Guide: Basic OS*

taskShow()

NAME *taskShow()* – display task information from TCBs

SYNOPSIS `STATUS taskShow
(
 int tid, /* task ID */
 int level /* 0 = summary, 1 = details, 2 = all tasks */
)`

DESCRIPTION This routine displays the contents of a task control block (TCB) for a specified task. If *level* is 1, it also displays task options and registers. If *level* is 2, it displays all tasks.

The TCB display contains the following fields:

| Field | Meaning |
|--------------|--|
| NAME | Task name |
| ENTRY | Symbol name or address where task began execution |
| TID | Task ID |
| PRI | Priority |
| STATUS | Task status, as formatted by <i>taskStatusString()</i> |
| PC | Program counter |

| Field | Meaning |
|-------|--|
| SP | Stack pointer |
| ERRNO | Most recent error code for this task |
| DELAY | If task is delayed, number of clock ticks remaining in delay (0 otherwise) |

EXAMPLE The following example shows the TCB contents for the shell task:

```
-> taskShow tShell, 1
  NAME          ENTRY      TID      PRI  STATUS      PC          SP          ERRNO  DELAY
-----
tShell         _shell        20efcac   1  READY      201dc90    20ef980      0       0
stack: base 0x20efcac end 0x20ed59c size 9532 high 1452 margin 8080
options: 0x1e
VX_UNBREAKABLE      VX_DEALLOC_STACK      VX_FP_TASK              VX_STDIO
D0 =      0   D4 =      0   A0 =      0   A4 =      0
D1 =      0   D5 =      0   A1 =      0   A5 = 203a084   SR =      3000
D2 =      0   D6 =      0   A2 =      0   A6 = 20ef9a0   PC = 2038614
D3 =      0   D7 =      0   A3 =      0   A7 = 20ef980
```

RETURNS N/A

SEE ALSO **taskShow**, **taskStatusString()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

taskShowInit()

NAME **taskShowInit()** – initialize the task show routine facility

SYNOPSIS `void taskShowInit (void)`

DESCRIPTION This routine links the task show routines into the VxWorks system. It is called automatically when the task show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define **INCLUDE_SHOW_ROUTINES** in **config.h**.
- If you use the Tornado project facility, select **INCLUDE_TASK_SHOW**.

RETURNS N/A

SEE ALSO **taskShow**

taskSpawn()

taskSpawn()**NAME** *taskSpawn()* – spawn a task

SYNOPSIS

```
int taskSpawn
(
    char * name,          /* name of new task (stored at pStackBase) */
    int priority,        /* priority of new task */
    int options,         /* task option word */
    int stackSize,       /* size (bytes) of stack needed plus name */
    FUNCPTR entryPt,     /* entry point of new task */
    int arg1,            /* 1st of 10 req'd task args to pass to func */
    int arg2,
    int arg3,
    int arg4,
    int arg5,
    int arg6,
    int arg7,
    int arg8,
    int arg9,
    int arg10
)
```

DESCRIPTION This routine creates and activates a new task with a specified priority and options and returns a system-assigned ID. See *taskInit()* and *taskActivate()* for the building blocks of this routine.

A task may be assigned a name as a debugging aid. This name will appear in displays generated by various system information facilities such as *i()*. The name may be of arbitrary length and content, but the current VxWorks convention is to limit task names to ten characters and prefix them with a "t". If *name* is specified as NULL, an ASCII name will be assigned to the task of the form "tn" where *n* is an integer which increments as new tasks are spawned.

The only resource allocated to a spawned task is a stack of a specified size *stackSize*, which is allocated from the system memory partition. Stack size should be an even integer. A task control block (TCB) is carved from the stack, as well as any memory required by the task name. The remaining memory is the task's stack and every byte is filled with the value 0xEE for the *checkStack()* facility. See the manual entry for *checkStack()* for stack-size checking aids.

The entry address *entryPt* is the address of the "main" routine of the task. The routine will be called once the C environment has been set up. The specified routine will be called with the ten given arguments. Should the specified main routine return, a call to *exit()* will automatically be made.

Note that ten (and only ten) arguments must be passed for the spawned function.

Bits in the options argument may be set to run with the following modes:

- VX_FP_TASK** (0x0008)
execute with floating-point coprocessor support.
- VX_PRIVATE_ENV** (0x0080)
include private environment support (see **envLib**).
- VX_NO_STACK_FILL** (0x0100)
do not fill the stack for use by *checkStack()*.
- VX_UNBREAKABLE** (0x0002)
do not allow breakpoint debugging.

See the definitions in **taskLib.h**.

- RETURNS** The task ID, or ERROR if memory is insufficient or the task cannot be created.
- ERRNO** **S_intLib_NOT_ISR_CALLABLE**, **S_objLib_OBJ_ID_ERROR**, **S_smObjLib_NOT_INITIALIZED**, **S_memLib_NOT_ENOUGH_MEMORY**, **S_memLib_BLOCK_ERROR**
- SEE ALSO** **taskLib**, *taskInit()*, *taskActivate()*, *sp()*, *VxWorks Programmer's Guide: Basic OS*

taskSRInit()

- NAME** *taskSRInit()* – initialize the default task status register (MIPS)
- SYNOPSIS**
- ```

ULONG taskSRInit
(
 ULONG newSRValue /* new default task status register */
)

```
- DESCRIPTION** This routine sets the default status register for system-wide tasks. All tasks will be spawned with the status register set to this value; thus, it must be called before *kernelInit()*.
- RETURNS** The previous value of the default status register.
- SEE ALSO** **taskArchLib**

---

## ***taskSRSet()***

**NAME** *taskSRSet()* – set the task status register (MC680x0, MIPS, i386/i486)

**SYNOPSIS**

```
STATUS taskSRSet
(
 int tid, /* task ID */
 UINT16 sr /* new SR */
)
```

**DESCRIPTION** This routine sets the status register of a task that is not running (i.e., the TCB must not be that of the calling task). Debugging facilities use this routine to set the trace bit in the status register of a task that is being single-stepped.

**RETURNS** OK, or ERROR if the task ID is invalid.

**SEE ALSO** *taskArchLib*

---

## ***taskStatusString()***

**NAME** *taskStatusString()* – get a task's status as a string

**SYNOPSIS**

```
STATUS taskStatusString
(
 int tid, /* task to get string for */
 char * pString /* where to return string */
)
```

**DESCRIPTION** This routine deciphers the WIND task status word in the TCB for a specified task, and copies the appropriate string to *pString*.

The formatted string is one of the following:

| <b>String</b> | <b>Meaning</b>                                                             |
|---------------|----------------------------------------------------------------------------|
| READY         | Task is not waiting for any resource other than the CPU.                   |
| PEND          | Task is blocked due to the unavailability of some resource.                |
| DELAY         | Task is asleep for some duration.                                          |
| SUSPEND       | Task is unavailable for execution (but not suspended, delayed, or pended). |
| DELAY+S       | Task is both delayed and suspended.                                        |
| PEND+S        | Task is both pended and suspended.                                         |
| PEND+T        | Task is pended with a timeout.                                             |

| String   | Meaning                                                               |
|----------|-----------------------------------------------------------------------|
| PEND+S+T | Task is pended with a timeout, and also suspended.                    |
| ...+I    | Task has inherited priority (+I may be appended to any string above). |
| DEAD     | Task no longer exists.                                                |

**EXAMPLE**

```
-> taskStatusString (taskNameToId ("tShell"), xx=malloc (10))
new symbol "xx" added to symbol table.
-> printf ("shell status = <%s>\n", xx)
shell status = <READY>
```

**RETURNS** OK, or ERROR if the task ID is invalid.

**SEE ALSO** `taskShow`

---

## *taskSuspend()*

**NAME** `taskSuspend()` – suspend a task

**SYNOPSIS**

```
STATUS taskSuspend
(
 int tid /* task ID of task to suspend */
)
```

**DESCRIPTION** This routine suspends a specified task. A task ID of zero results in the suspension of the calling task. Suspension is additive, thus tasks can be delayed and suspended, or pended and suspended. Suspended, delayed tasks whose delays expire remain suspended. Likewise, suspended, pended tasks that unblock remain suspended only.

Care should be taken with asynchronous use of this facility. The specified task is suspended regardless of its current state. The task could, for instance, have mutual exclusion to some system resource, such as the network \* or system memory partition. If suspended during such a time, the facilities engaged are unavailable, and the situation often ends in deadlock.

This routine is the basis of the debugging and exception handling packages. However, as a synchronization mechanism, this facility should be rejected in favor of the more general semaphore facility.

**RETURNS** OK, or ERROR if the task cannot be suspended.

**ERRNO** `S_objLib_OBJ_ID_ERROR`

**SEE ALSO** `taskLib`

---

## *taskSwitchHookAdd()*

**NAME** *taskSwitchHookAdd()* – add a routine to be called at every task switch

**SYNOPSIS**

```
STATUS taskSwitchHookAdd
(
 FUNCPTR switchHook /* routine to be called at every task switch */
)
```

**DESCRIPTION** This routine adds a specified routine to a list of routines that will be called at every task switch. The routine should be declared as follows:

```
void switchHook
(
 WIND_TCB *pOldTcb, /* pointer to old task's WIND_TCB */
 WIND_TCB *pNewTcb /* pointer to new task's WIND_TCB */
)
```

**NOTE** User-installed switch hooks are called within the kernel context. Therefore, switch hooks do not have access to all VxWorks facilities. The following routines can be called from within a task switch hook:

| <b>Library</b>    | <b>Routines</b>                                                                                                        |
|-------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>bLib</b>       | All routines                                                                                                           |
| <b>fppArchLib</b> | <i>fppSave()</i> , <i>fppRestore()</i>                                                                                 |
| <b>intLib</b>     | <i>intContext()</i> , <i>intCount()</i> , <i>intVecSet()</i> , <i>intVecGet()</i>                                      |
| <b>lstLib</b>     | All routines                                                                                                           |
| <b>mathALib</b>   | All routines, if <i>fppSave()</i> / <i>fppRestore()</i> are used                                                       |
| <b>rngLib</b>     | All routines except <i>rngCreate()</i>                                                                                 |
| <b>taskLib</b>    | <i>taskIdVerify()</i> , <i>taskIdDefault()</i> , <i>taskIsReady()</i> ,<br><i>taskIsSuspended()</i> , <i>taskTcb()</i> |
| <b>vxLib</b>      | <i>vxTas()</i>                                                                                                         |

**RETURNS** OK, or ERROR if the table of task switch routines is full.

**SEE ALSO** *taskHookLib*, *taskSwitchHookDelete()*

---

## *taskSwitchHookDelete()*

**NAME** *taskSwitchHookDelete()* – delete a previously added task switch routine

**SYNOPSIS**

```
STATUS taskSwitchHookDelete
(
 FUNCPTR switchHook /* routine to be deleted from list */
)
```

**DESCRIPTION** This routine removes the specified routine from the list of routines to be called at each task switch.

**RETURNS** OK, or ERROR if the routine is not in the table of task switch routines.

**SEE ALSO** *taskHookLib, taskSwitchHookAdd()*

---

## *taskSwitchHookShow()*

**NAME** *taskSwitchHookShow()* – show the list of task switch routines

**SYNOPSIS**

```
void taskSwitchHookShow (void)
```

**DESCRIPTION** This routine shows all the switch routines installed in the task switch hook table, in the order in which they were installed.

**RETURNS** N/A

**SEE ALSO** *taskHookShow, taskSwitchHookAdd()*

---

## *taskTcb()*

**NAME** *taskTcb()* – get the task control block for a task ID

**SYNOPSIS**

```
WIND_TCB *taskTcb
(
 int tid /* task ID */
)
```

***taskUnlock()***

|                    |                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine returns a pointer to the task control block ( <b>WIND_TCB</b> ) for a specified task. Although all task state information is contained in the TCB, users must not modify it directly. To change registers, for instance, use <i>taskRegsSet()</i> and <i>taskRegsGet()</i> . |
| <b>RETURNS</b>     | A pointer to a <b>WIND_TCB</b> , or <b>NULL</b> if the task ID is invalid.                                                                                                                                                                                                                |
| <b>ERRNO</b>       | <b>S_objLib_OBJ_ID_ERROR</b>                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <b>taskLib</b>                                                                                                                                                                                                                                                                            |

---

***taskUnlock()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>taskUnlock()</i> – enable task rescheduling                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <b>STATUS</b> <i>taskUnlock</i> (void)                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | This routine decrements the preemption lock count. Typically this call is paired with <i>taskLock()</i> and concludes a critical section of code. Preemption will not be unlocked until <i>taskUnlock()</i> has been called as many times as <i>taskLock()</i> . When the lock count is decremented to zero, any tasks that were eligible to preempt the current task will execute. The <i>taskUnlock()</i> routine is not callable from interrupt service routines. |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>ERRNO</b>       | <b>S_intLib_NOT_ISR_CALLABLE</b>                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <b>taskLib</b> , <i>taskLock()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                   |

---

***taskUnsafe()***

|                    |                                                                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>taskUnsafe()</i> – make the calling task unsafe from deletion                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <b>STATUS</b> <i>taskUnsafe</i> (void)                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | This routine removes the calling task's protection from deletion. Tasks that attempt to delete a protected task will block until the task is unsafe. When a task becomes unsafe, the deleter will be unblocked and allowed to delete the task. |

The *taskUnsafe()* primitive utilizes a count to keep track of nested calls for task protection. When nesting occurs, the task becomes unsafe only after the outermost *taskUnsafe()* is executed.

**RETURNS** OK.

**SEE ALSO** *taskLib*, *taskSafe()*, *VxWorks Programmer's Guide: Basic OS*

---

## *taskVarAdd()*

**NAME** *taskVarAdd()* – add a task variable to a task

**SYNOPSIS**

```
STATUS taskVarAdd
(
 int tid, /* ID of task to have new variable */
 int * pVar /* pointer to variable to be switched for task */
)
```

**DESCRIPTION** This routine adds a specified variable *pVar* (4-byte memory location) to a specified task's context. After calling this routine, the variable will be private to the task. The task can access and modify the variable, but the modifications will not appear to other tasks, and other tasks' modifications to that variable will not affect the value seen by the task. This is accomplished by saving and restoring the variable's initial value each time a task switch occurs to or from the calling task.

This facility can be used when a routine is to be spawned repeatedly as several independent tasks. Although each task will have its own stack, and thus separate stack variables, they will all share the same static and global variables. To make a variable *not* shareable, the routine can call *taskVarAdd()* to make a separate copy of the variable for each task, but all at the same physical address.

Note that task variables increase the task switch time to and from the tasks that own them. Therefore, it is desirable to limit the number of task variables that a task uses. One efficient way to use task variables is to have a single task variable that is a pointer to a dynamically allocated structure containing the task's private data.

**EXAMPLE** Assume that three identical tasks were spawned with a routine called *operator()*. All three use the structure **OP\_GLOBAL** for all variables that are specific to a particular incarnation of the task. The following code fragment shows how this is set up:

***taskVarDelete()***

```

OP_GLOBAL *opGlobal; /* ptr to operator task's global variables */
void operator
(
 int opNum /* number of this operator task */
)
{
 if (taskVarAdd (0, (int *)&opGlobal) != OK)
 {
 printErr ("operator%d: can't taskVarAdd opGlobal\n", opNum);
 taskSuspend (0);
 }
 if ((opGlobal = (OP_GLOBAL *) malloc (sizeof (OP_GLOBAL))) == NULL)
 {
 printErr ("operator%d: can't malloc opGlobal\n", opNum);
 taskSuspend (0);
 }
 ...
}

```

**RETURNS** OK, or ERROR if memory is insufficient for the task variable descriptor.

**SEE ALSO** *taskVarLib*, *taskVarDelete()*, *taskVarGet()*, *taskVarSet()*

***taskVarDelete()***

**NAME** *taskVarDelete()* – remove a task variable from a task

**SYNOPSIS** STATUS *taskVarDelete*

```

(
 int tid, /* ID of task whose variable is to be removed */
 int * pVar /* pointer to task variable to be removed */
)

```

**DESCRIPTION** This routine removes a specified task variable, *pVar*, from the specified task's context. The private value of that variable is lost.

**RETURNS** OK, or ERROR if the task variable does not exist for the specified task.

**SEE ALSO** *taskVarLib*, *taskVarAdd()*, *taskVarGet()*, *taskVarSet()*



---

## *taskVarGet()*

|                    |                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>taskVarGet()</i> – get the value of a task variable                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>int taskVarGet (     int    tid, /* ID of task whose task variable is to be retrieved */     int * pVar /* pointer to task variable */ )</pre>                                                                                                                |
| <b>DESCRIPTION</b> | This routine returns the private value of a task variable for a specified task. The specified task is usually not the calling task, which can get its private value by directly accessing the variable. This routine is provided primarily for debugging purposes. |
| <b>RETURNS</b>     | The private value of the task variable, or ERROR if the task is not found or it does not own the task variable.                                                                                                                                                    |
| <b>SEE ALSO</b>    | <i>taskVarLib</i> , <i>taskVarAdd()</i> , <i>taskVarDelete()</i> , <i>taskVarSet()</i>                                                                                                                                                                             |

---

## *taskVarInfo()*

|                    |                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>taskVarInfo()</i> – get a list of task variables of a task                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>int taskVarInfo (     int    tid,          /* ID of task whose task variable is to be set */     TASK_VAR varList[], /* array to hold task variable addresses */     int    maxVars      /* maximum variables varList can accommodate */ )</pre> |
| <b>DESCRIPTION</b> | This routine provides the calling task with a list of all of the task variables of a specified task. The unsorted array of task variables is copied to <i>varList</i> .                                                                               |
| <b>CAVEATS</b>     | Kernel rescheduling is disabled with <i>taskLock()</i> while task variables are looked up. There is no guarantee that all the task variables are still valid or that new task variables have not been created by the time this routine returns.       |
| <b>RETURNS</b>     | The number of task variables in the list.                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <i>taskVarLib</i>                                                                                                                                                                                                                                     |

**taskVarInit()**

---

## *taskVarInit()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>taskVarInit()</i> – initialize the task variables facility                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <b>STATUS</b> <code>taskVarInit (void)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This routine initializes the task variables facility. It installs task switch and delete hooks used for implementing task variables. If <i>taskVarInit()</i> is not called explicitly, <i>taskVarAdd()</i> will call it automatically when the first task variable is added.</p> <p>After the first invocation of this routine, subsequent invocations have no effect.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>WARNING</b>     | <p>Order dependencies in task delete hooks often involve task variables. If a facility uses task variables and has a task delete hook that expects to use those task variables, the facility's delete hook must run before the task variables' delete hook. Otherwise, the task variables will be deleted by the time the facility's delete hook runs.</p> <p>VxWorks is careful to run the delete hooks in reverse of the order in which they were installed. Any facility that has a delete hook that will use task variables can guarantee proper ordering by calling <i>taskVarInit()</i> before adding its own delete hook.</p> <p>Note that this is not an issue in normal use of task variables. The issue only arises when adding another task delete hook that uses task variables.</p> <p>Caution should also be taken when adding task variables from within create hooks. If the task variable package has not been installed via <i>taskVarInit()</i>, the create hook attempts to create a create hook, and that may cause system failure. To avoid this situation, <i>taskVarInit()</i> should be called during system initialization from the root task, <i>usrRoot()</i>, in <i>usrConfig.c</i>.</p> |
| <b>RETURNS</b>     | OK, or ERROR if the task switch/delete hooks could not be installed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <i>taskVarLib</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

## *taskVarSet()*

|                    |                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>taskVarSet()</i> – set the value of a task variable                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS taskVarSet (     int    tid, /* ID of task whose task variable is to be set */     int * pVar, /* pointer to task variable to be set for this task */     int    value /* new value of task variable */ )</pre>                                       |
| <b>DESCRIPTION</b> | This routine sets the private value of the task variable for a specified task. The specified task is usually not the calling task, which can set its private value by directly modifying the variable. This routine is provided primarily for debugging purposes. |
| <b>RETURNS</b>     | OK, or ERROR if the task is not found or it does not own the task variable.                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <i>taskVarLib</i> , <i>taskVarAdd()</i> , <i>taskVarDelete()</i> , <i>taskVarGet()</i>                                                                                                                                                                            |

---

## *tcicInit()*

|                    |                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tcicInit()</i> – initialize the TCIC chip                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS tcicInit (     int    ioBase, /* IO base address */     int    intVec, /* interrupt vector */     int    intLevel, /* interrupt level */     FUNCPTR showRtn /* show routine */ )</pre> |
| <b>DESCRIPTION</b> | This routine initializes the TCIC chip.                                                                                                                                                             |
| <b>RETURNS</b>     | OK, or ERROR if the TCIC chip cannot be found.                                                                                                                                                      |
| <b>SEE ALSO</b>    | <i>tcic</i>                                                                                                                                                                                         |

## ***tcicShow()***

**NAME** *tcicShow()* – show all configurations of the TCIC chip

**SYNOPSIS**

```
void tcicShow
(
 int sock /* socket no. */
)
```

**DESCRIPTION** This routine shows all configurations of the TCIC chip.

**RETURNS** N/A

**SEE ALSO** *tcicShow*

---

## ***tcpDebugShow()***

**NAME** *tcpDebugShow()* – display debugging information for the TCP protocol

**SYNOPSIS**

```
void tcpDebugShow
(
 int numPrint, /* no. of entries to print, default (0) = 20 */
 int verbose /* 1 = verbose */
)
```

**DESCRIPTION** This routine displays debugging information for the TCP protocol. To include TCP debugging facilities, define `INCLUDE_TCP_DEBUG` when building the system image. To enable information gathering, turn on the `SO_DEBUG` option for the relevant socket(s).

**RETURNS** N/A

**SEE ALSO** *tcpShow*

---

## *tcpShowInit()*

|                    |                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tcpShowInit()</i> – initialize TCP show routines                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <code>void tcpShowInit (void)</code>                                                                                                                                                                                |
| <b>DESCRIPTION</b> | This routine links the TCP show facility into the VxWorks system. These routines are included automatically if <code>INCLUDE_NET_SHOW</code> and <code>INCLUDE_TCP</code> are defined in <code>configAll.h</code> . |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <code>tcpShow</code>                                                                                                                                                                                                |

---

## *tcpstatShow()*

|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <b>NAME</b>        | <i>tcpstatShow()</i> – display all statistics for the TCP protocol |
| <b>SYNOPSIS</b>    | <code>void tcpstatShow (void)</code>                               |
| <b>DESCRIPTION</b> | This routine displays detailed statistics for the TCP protocol.    |
| <b>RETURNS</b>     | N/A                                                                |
| <b>SEE ALSO</b>    | <code>tcpShow</code>                                               |

---

## *tcw()*

|                    |                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tcw()</i> – return the contents of the <code>tcw</code> register (i960)                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>int tcw (     int taskId /* task ID, 0 means default task */ )</pre>                                                                                                    |
| <b>DESCRIPTION</b> | This command extracts the contents of the <code>tcw</code> register from the TCB of a specified task. If <i>taskId</i> is omitted or 0, the current default task is assumed. |

## ***td()***

- RETURNS** The contents of the **tcw** register.
- SEE ALSO** **dbgArchLib**, *VxWorks Programmer's Guide: Target Shell*

---

## ***td()***

- NAME** *td()* – delete a task
- SYNOPSIS**
- ```
void td
(
    int taskNameOrId /* task name or task ID */
)
```
- DESCRIPTION** This command deletes a specified task. It simply calls *taskDelete()*.
- RETURNS** N/A
- SEE ALSO** **usrLib**, *taskDelete()*, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

telnetd()

- NAME** *telnetd()* – VxWorks telnet daemon
- SYNOPSIS**
- ```
void telnetd (void)
```
- DESCRIPTION** This routine enables remote users to log in to VxWorks over the network via the telnet protocol. It is spawned by *telnetInit()*, which should be called at boot time.
- Remote telnet requests will cause **stdin**, **stdout**, and **stderr** to be stolen away from the console. When the remote user disconnects, **stdin**, **stdout**, and **stderr** are restored, and the shell is restarted.
- The telnet daemon requires the existence of a pseudo-terminal device, which is created by *telnetInit()* before *telnetd()* is spawned. The *telnetd()* routine creates two additional processes, **tTelnetInTask** and **tTelnetOutTask**, whenever a remote user is logged in. These processes exit when the remote connection is terminated.
- RETURNS** N/A
- SEE ALSO** **telnetLib**

---

## *telnetInit()*

**NAME** *telnetInit()* – initialize the telnet daemon

**SYNOPSIS** `void telnetInit (void)`

**DESCRIPTION** This routine initializes the telnet facility, which supports remote login to the VxWorks shell via the telnet protocol. It creates a pty device and spawns the telnet daemon. It is called automatically when the configuration macro `INCLUDE_TELNET` is defined.

**RETURNS** N/A

**SEE ALSO** `telnetLib`

---

## *tftpCopy()*

**NAME** *tftpCopy()* – transfer a file via TFTP

**SYNOPSIS**

```
STATUS tftpCopy
(
 char * pHost, /* host name or address */
 int port, /* optional port number */
 char * pFilename, /* remote filename */
 char * pCommand, /* TFTP command */
 char * pMode, /* TFTP transfer mode */
 int fd /* fd to put/get data */
)
```

**DESCRIPTION** This routine transfers a file using the TFTP protocol to or from a remote system. *pHost* is the remote server name or Internet address. A non-zero value for *port* specifies an alternate TFTP server port (zero means use default TFTP port number (69)). *pFilename* is the remote filename. *pCommand* specifies the TFTP command, which can be either "put" or "get". *pMode* specifies the mode of transfer, which can be "ascii", "netascii", "binary", "image", or "octet".

*fd* is a file descriptor from which to read/write the data from or to the remote system. For example, if the command is "get", the remote data will be written to *fd*. If the command is "put", the data to be sent is read from *fd*. The caller is responsible for managing *fd*. That is, *fd* must be opened prior to calling *tftpCopy()* and closed up on completion.

***tftpdDirectoryAdd()***

**EXAMPLE** The following sequence gets an ASCII file "/folk/vw/xx.yy" on host "congo" and stores it to a local file called "localfile":

```
-> fd = open ("localfile", 0x201, 0644)
-> tftpdCopy ("congo", 0, "/folk/vw/xx.yy", "get", "ascii", fd)
-> close (fd)
```

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_tftpdLib\_INVALID\_COMMAND

**SEE ALSO** tftpdLib, ftpLib

---

***tftpdDirectoryAdd()***

**NAME** *tftpdDirectoryAdd()* – add a directory to the access list

**SYNOPSIS** STATUS tftpdDirectoryAdd  
(  
char \* fileName /\* name of directory to add to access list \*/  
)

**DESCRIPTION** This routine adds the specified directory name to the access list for the TFTP server.

**RETURNS** N/A

**SEE ALSO** tftpdLib

---

***tftpdDirectoryRemove()***

**NAME** *tftpdDirectoryRemove()* – delete a directory from the access list

**SYNOPSIS** STATUS tftpdDirectoryRemove  
(  
char \* fileName /\* name of directory to add to access list \*/  
)

**DESCRIPTION** This routine deletes the specified directory name from the access list for the TFTP server.



RETURNS N/A

SEE ALSO *tftpdLib*

---

## *tftpdInit()*

NAME *tftpdInit()* – initialize the TFTP server task

SYNOPSIS

```
STATUS tftpdInit
(
 int stackSize, /* stack size for the tftpdTask */
 int nDirectories, /* number of directories allowed read */
 char * *directoryNames, /* array of dir names */
 BOOL noControl, /* TRUE if no access control required */
 int maxConnections
)
```

DESCRIPTION This routine will spawn a new TFTP server task, if one does not already exist. If a TFTP server task is running already, *tftpdInit()* will simply return without creating a new task. It will simply report whether a new TFTP task was successfully spawned. The argument *stackSize* can be specified to change the default stack size for the TFTP server task. The default size is set in the global variable *tftpdTaskStackSize*.

RETURNS OK, or ERROR if a new TFTP task cannot be created.

SEE ALSO *tftpdLib*

---

## *tftpdTask()*

NAME *tftpdTask()* – TFTP server daemon task

SYNOPSIS

```
STATUS tftpdTask
(
 int nDirectories, /* number of dirs allowed access */
 char * *directoryNames, /* array of directory names */
 int maxConnections /* max number of simultan. connects */
)
```

***tftpGet()***

|                    |                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine processes incoming TFTP client requests by spawning a new task for each connection that is set up.<br><br>This routine is called by <i>tftpdInit()</i> . |
| <b>RETURNS</b>     | OK, or ERROR if the task returns unexpectedly.                                                                                                                        |
| <b>SEE ALSO</b>    | <i>tftpdLib</i>                                                                                                                                                       |

---

***tftpGet()***

|                    |                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tftpGet()</i> – get a file from a remote system                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre> STATUS tftpGet (     TFTP_DESC * pTftpDesc,    /* TFTP descriptor */     char *     pFilename,    /* remote filename */     int       fd,            /* file descriptor */     int       clientOrServer /* which side is calling */ ) </pre>                                            |
| <b>DESCRIPTION</b> | This routine gets a file from a remote system via TFTP. <i>pFilename</i> is the filename. <i>fd</i> is the file descriptor to which the data is written. <i>pTftpDesc</i> is a pointer to the TFTP descriptor. The <i>tftpPeerSet()</i> routine must be called prior to calling this routine. |
| <b>RETURNS</b>     | OK, or ERROR if unsuccessful.                                                                                                                                                                                                                                                                 |
| <b>ERRNO</b>       | S_tftpLib_INVALID_DESCRIPTOR<br>S_tftpLib_INVALID_ARGUMENT<br>S_tftpLib_NOT_CONNECTED                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <i>tftpLib</i>                                                                                                                                                                                                                                                                                |

---

## ***tftpInfoShow()***

|                    |                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tftpInfoShow()</i> – get TFTP status information                                                                                                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>STATUS tftpInfoShow (     TFTP_DESC * pTftpDesc /* TFTP descriptor */ )</pre>                                                                                                                                                                    |
| <b>DESCRIPTION</b> | This routine prints information associated with TFTP descriptor <i>pTftpDesc</i> .                                                                                                                                                                    |
| <b>EXAMPLE</b>     | A call to <i>tftpInfoShow()</i> might look like:<br><pre>-&gt; tftpInfoShow (tftpDesc)     Connected to yuba [69]     Mode: netascii Verbose: off Tracing: off     Rexmt-interval: 5 seconds, Max-timeout: 25 seconds     value = 0 = 0x0 -&gt;</pre> |
| <b>RETURNS</b>     | OK, or ERROR if unsuccessful.                                                                                                                                                                                                                         |
| <b>ERRNO</b>       | S_tftpLib_INVALID_DESCRIPTOR                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>tftpLib</b>                                                                                                                                                                                                                                        |

---

## ***tftpInit()***

|                    |                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tftpInit()</i> – initialize a TFTP session                                                                                              |
| <b>SYNOPSIS</b>    | <pre>TFTP_DESC * tftpInit (void)</pre>                                                                                                     |
| <b>DESCRIPTION</b> | This routine initializes a TFTP session by allocating and initializing a TFTP descriptor. It sets the default transfer mode to "netascii". |
| <b>RETURNS</b>     | A pointer to a TFTP descriptor if successful, otherwise NULL.                                                                              |
| <b>SEE ALSO</b>    | <b>tftpLib</b>                                                                                                                             |

---

## ***tftpModeSet()***

**NAME** *tftpModeSet()* – set the TFTP transfer mode

**SYNOPSIS**

```
STATUS tftpModeSet
(
 TFTP_DESC * pTftpDesc, /* TFTP descriptor */
 char * pMode /* TFTP transfer mode */
)
```

**DESCRIPTION** This routine sets the transfer mode associated with the TFTP descriptor *pTftpDesc*. *pMode* specifies the transfer mode, which can be "netascii", "binary", "image", or "octet". Although recognized, these modes actually translate into either octet or netascii.

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_tftpLib\_INVALID\_DESCRIPTOR  
S\_tftpLib\_INVALID\_ARGUMENT  
S\_tftpLib\_INVALID\_MODE

**SEE ALSO** *tftpLib*

---

## ***tftpPeerSet()***

**NAME** *tftpPeerSet()* – set the TFTP server address

**SYNOPSIS**

```
STATUS tftpPeerSet
(
 TFTP_DESC * pTftpDesc, /* TFTP descriptor */
 char * pHostname, /* server name/address */
 int port /* port number */
)
```

**DESCRIPTION** This routine sets the TFTP server (peer) address associated with the TFTP descriptor *pTftpDesc*. *pHostname* is either the TFTP server name (e.g., "congo") or the server Internet address (e.g., "90.3"). A non-zero value for *port* specifies the server port number (zero means use the default TFTP server port number (69)).

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_tftpLib\_INVALID\_DESCRIPTOR  
S\_tftpLib\_INVALID\_ARGUMENT  
S\_tftpLib\_UNKNOWN\_HOST

**SEE ALSO** tftpLib

---

## *tftpPut()*

**NAME** *tftpPut()* – put a file to a remote system

**SYNOPSIS**

```
STATUS tftpPut
(
 TFTP_DESC * pTftpDesc, /* TFTP descriptor */
 char * pFilename, /* remote filename */
 int fd, /* file descriptor */
 int clientOrServer /* which side is calling */
)
```

**DESCRIPTION** This routine puts data from a local file (descriptor) to a file on the remote system. *pTftpDesc* is a pointer to the TFTP descriptor. *pFilename* is the remote filename. *fd* is the file descriptor from which it gets the data. A call to *tftpPeerSet()* must be made prior to calling this routine.

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_tftpLib\_INVALID\_DESCRIPTOR  
S\_tftpLib\_INVALID\_ARGUMENT  
S\_tftpLib\_NOT\_CONNECTED

**SEE ALSO** tftpLib

---

## *tftpQuit()*

**NAME** *tftpQuit()* – quit a TFTP session

**SYNOPSIS**

```
STATUS tftpQuit
(
 TFTP_DESC * pTftpDesc /* TFTP descriptor */
)
```

|                    |                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine closes a TFTP session associated with the TFTP descriptor <i>pTftpDesc</i> . |
| <b>RETURNS</b>     | OK, or ERROR if unsuccessful.                                                             |
| <b>ERRNO</b>       | S_tftpLib_INVALID_DESCRIPTOR                                                              |
| <b>SEE ALSO</b>    | tftpLib                                                                                   |

---

## **tftpSend()**

**NAME** *tftpSend()* – send a TFTP message to the remote system

**SYNOPSIS**

```
int tftpSend
(
 TFTP_DESC * pTftpDesc, /* TFTP descriptor */
 TFTP_MSG * pTftpMsg, /* TFTP send message */
 int sizeMsg, /* send message size */
 TFTP_MSG * pTftpReply, /* TFTP reply message */
 int opReply, /* reply opcode */
 int blockReply, /* reply block number */
 int * pPort /* return port number */
)
```

**DESCRIPTION** This routine sends *sizeMsg* bytes of the passed message *pTftpMsg* to the remote system associated with the TFTP descriptor *pTftpDesc*. If *pTftpReply* is not NULL, **tftpSend()** tries to get a reply message with a block number *blockReply* and an opcode *opReply*. If *pPort* is NULL, the reply message must come from the same port to which the message was sent. If *pPort* is not NULL, the port number from which the reply message comes is copied to this variable.

**RETURNS** The size of the reply message, or ERROR.

**ERRNO** S\_tftpLib\_TIMED\_OUT  
S\_tftpLib\_TFTP\_ERROR

**SEE ALSO** tftpLib

---

## *tftpXfer()*

**NAME** *tftpXfer()* – transfer a file via TFTP using a stream interface

**SYNOPSIS**

```
STATUS tftpXfer
(
 char * pHost, /* host name or address */
 int port, /* port number */
 char * pFilename, /* remote filename */
 char * pCommand, /* TFTP command */
 char * pMode, /* TFTP transfer mode */
 int * pDataDesc, /* return data desc. */
 int * pErrorDesc /* return error desc. */
)
```

**DESCRIPTION** This routine initiates a transfer to or from a remote file via TFTP. It spawns a task to perform the TFTP transfer and returns a descriptor from which the data can be read (for "get") or to which it can be written (for "put") interactively. The interface for this routine is similar to *ftpXfer()* in **ftpLib**.

*pHost* is the server name or Internet address. A non-zero value for *ports* specifies an alternate TFTP server port number (zero means use default TFTP port number (69)). *pFilename* is the remote filename. *pCommand* specifies the TFTP command. The command can be either "put" or "get".

The *tftpXfer()* routine returns a data descriptor, in *pDataDesc*, from which the TFTP data is read (for "get") or to which it is written (for "put"). An error status descriptor gets returned in the variable *pErrorDesc*. If an error occurs during the TFTP transfer, an error string can be read from this descriptor. After returning successfully from *tftpXfer()*, the calling application is responsible for closing both descriptors.

If there are delays in reading or writing the data descriptor, it is possible for the TFTP transfer to time out.

**EXAMPLE** The following code demonstrates how *tftpXfer()* may be used:

```
#include "tftpLib.h"
#define BUFFERSIZE 512
int dataFd;
int errorFd;
int num;
char buf [BUFFERSIZE + 1];
if (tftpXfer ("congo", 0, "/usr/fred", "get", "ascii", &dataFd,
 &errorFd) == ERROR)
 return (ERROR);
while ((num = read (dataFd, buf, sizeof (buf))) > 0)
```

**ti()**

```

 {

 }
 close (dataFd);
 num = read (errorFd, buf, BUFFERSIZE);
 if (num > 0)
 {
 buf [num] = '\0';
 printf ("YIKES! An error occurred!:%s\n", buf);

 }
 close (errorFd);

```

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_ftpLib\_INVALID\_ARGUMENT

**SEE ALSO** tftpLib, ftpLib

**ti()**

**NAME** *ti()* – print complete information from a task’s TCB

**SYNOPSIS**

```

void ti
(
 int taskNameOrId /* task name or task ID; 0 = use default */
)

```

**DESCRIPTION** This command prints the task control block (TCB) contents, including registers, for a specified task. If *taskNameOrId* is omitted or zero, the last task referenced is assumed.

The *ti()* routine uses *taskShow()*; see the documentation for *taskShow()* for a description of the output format.

**EXAMPLE** The following shows the TCB contents for the shell task:

```

-> ti
 NAME ENTRY TID PRI STATUS PC SP ERRNO DELAY

tShell _shell 20efcac 1 READY 201dc90 20ef980 0 0
stack: base 0x20efcac end 0x20ed59c size 9532 high 1452 margin 8080
options: 0x1e
VX_UNBREAKABLE VX_DEALLOC_STACK VX_FP_TASK VX_STDIO

```



```

D0 = 0 D4 = 0 A0 = 0 A4 = 0
D1 = 0 D5 = 0 A1 = 0 A5 = 203a084 SR = 3000
D2 = 0 D6 = 0 A2 = 0 A6 = 20ef9a0 PC = 2038614
D3 = 0 D7 = 0 A3 = 0 A7 = 20ef980
value = 34536868 = 0x20efda4

```

**RETURNS** N/A**SEE ALSO** *usrLib, taskShow()*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

---

## *tickAnnounce()*

**NAME** *tickAnnounce()* – announce a clock tick to the kernel**SYNOPSIS** `void tickAnnounce (void)`**DESCRIPTION** This routine informs the kernel of the passing of time. It should be called from an interrupt service routine that is connected to the system clock. The most common frequencies are 60Hz or 100Hz. Frequencies in excess of 600Hz are an inefficient use of processor power because the system will spend most of its time advancing the clock. By default, this routine is called by *usrClock()* in *usrConfig.c*.**RETURNS** N/A**SEE ALSO** *tickLib, kernelLib, taskLib, semLib, wdLib*, *VxWorks Programmer's Guide: Basic OS*

---

## *tickGet()*

**NAME** *tickGet()* – get the value of the kernel's tick counter**SYNOPSIS** `ULONG tickGet (void)`**DESCRIPTION** This routine returns the current value of the tick counter. This value is set to zero at startup, incremented by *tickAnnounce()*, and can be changed using *tickSet()*.**RETURNS** The most recent *tickSet()* value, plus all *tickAnnounce()* calls since.**SEE ALSO** *tickLib, tickSet(), tickAnnounce()*

**tickSet()**

---

**tickSet()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tickSet()</i> – set the value of the kernel’s tick counter                                                                                                                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>void tickSet (     ULONG ticks /* new time in ticks */ )</pre>                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | This routine sets the internal tick counter to a specified value in ticks. The new count will be reflected by <i>tickGet()</i> , but will not change any delay fields or timeouts selected for any tasks. For example, if a task is delayed for ten ticks, and this routine is called to advance time, the delayed task will still be delayed until ten <i>tickAnnounce()</i> calls have been made. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <i>tickLib</i> , <i>tickGet()</i> , <i>tickAnnounce()</i>                                                                                                                                                                                                                                                                                                                                           |

---

**time()**

|                      |                                                                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>time()</i> – determine the current calendar time (ANSI)                                                                                                                                                   |
| <b>SYNOPSIS</b>      | <pre>time_t time (     time_t * timer /* calendar time in seconds */ )</pre>                                                                                                                                 |
| <b>DESCRIPTION</b>   | This routine returns the implementation’s best approximation of current calendar time in seconds. If <i>timer</i> is non-NULL, the return value is also copied to the location to which <i>timer</i> points. |
| <b>INCLUDE FILES</b> | <i>time.h</i>                                                                                                                                                                                                |
| <b>RETURNS</b>       | The current calendar time in seconds, or ERROR (-1) if the calendar time is not available.                                                                                                                   |
| <b>SEE ALSO</b>      | <i>ansiTime</i> , <i>clock_gettime()</i>                                                                                                                                                                     |

---

## *timer\_cancel()*

|                    |                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>timer_cancel()</i> – cancel a timer                                                       |
| <b>SYNOPSIS</b>    | <pre>int timer_cancel (     timer_t timerid /* timer ID */ )</pre>                           |
| <b>DESCRIPTION</b> | This routine is a shorthand method of invoking <i>timer_settime()</i> , which stops a timer. |
| <b>NOTE</b>        | Non-POSIX.                                                                                   |
| <b>RETURNS</b>     | 0 (OK), or -1 (ERROR) if <i>timerid</i> is invalid.                                          |
| <b>ERRNO</b>       | EINVAL                                                                                       |
| <b>SEE ALSO</b>    | timerLib                                                                                     |

---

## *timer\_connect()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>timer_connect()</i> – connect a user routine to the timer signal                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>int timer_connect (     timer_t    timerid, /* timer ID */     VOIDFUNCPTR routine, /* user routine */     int        arg      /* user argument */ )</pre>                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This routine sets the specified <i>routine</i> to be invoked with <i>arg</i> when fielding a signal indicated by the timer's <i>evp</i> signal number, or if <i>evp</i> is NULL, when fielding the default signal (SIGALRM).</p> <p>The signal handling routine should be declared as:</p> <pre>void my_handler (     timer_t timerid, /* expired timer ID */     int arg        /* user argument */ )</pre> |

***timer\_create()***

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <b>NOTE</b>     | Non-POSIX.                                                                       |
| <b>RETURNS</b>  | 0 (OK), or -1 (ERROR) if the timer is invalid or cannot bind the signal handler. |
| <b>ERRNO</b>    | EINVAL                                                                           |
| <b>SEE ALSO</b> | <b>timerLib</b>                                                                  |

---

***timer\_create()***

**NAME** *timer\_create()* – allocate a timer using the specified clock for a timing base (POSIX)

**SYNOPSIS**

```
int timer_create
(
 clockid_t clock_id, /* clock ID (always CLOCK_REALTIME) */
 struct sigevent * evp, /* user event handler */
 timer_t * pTimer /* ptr to return value */
)
```

**DESCRIPTION** This routine returns a value in *pTimer* that identifies the timer in subsequent timer requests. The *evp* argument, if non-NULL, points to a **sigevent** structure, which is allocated by the application and defines the signal number and application-specific data to be sent to the task when the timer expires. If *evp* is NULL, a default signal (SIGALRM) is queued to the task, and the signal data is set to the timer ID. Initially, the timer is disarmed.

**RETURNS** 0 (OK), or -1 (ERROR) if too many timers already are allocated or the signal number is invalid.

**ERRNO** EMTIMERS, EINVAL, ENOSYS, EAGAIN, S\_memLib\_NOT\_ENOUGH\_MEMORY

**SEE ALSO** **timerLib**, *timer\_delete()*

---

## *timer\_delete()*

**NAME** *timer\_delete()* – remove a previously created timer (POSIX)

**SYNOPSIS**

```
int timer_delete
(
 timer_t timerid /* timer ID */
)
```

**DESCRIPTION** This routine removes a timer.

**RETURNS** 0 (OK), or -1 (ERROR) if *timerid* is invalid.

**ERRNO** EINVAL

**SEE ALSO** *timerLib*, *timer\_create()*

---

## *timer\_getoverrun()*

**NAME** *timer\_getoverrun()* – return the timer expiration overrun (POSIX)

**SYNOPSIS**

```
int timer_getoverrun
(
 timer_t timerid /* timer ID */
)
```

**DESCRIPTION** This routine returns the timer expiration overrun count for *timerid*, when called from a timer expiration signal catcher. The overrun count is the number of extra timer expirations that have occurred, up to the implementation-defined maximum `_POSIX_DELAYTIMER_MAX`. If the count is greater than the maximum, it returns the maximum.

**RETURNS** The number of overruns, or `_POSIX_DELAYTIMER_MAX` if the count equals or is greater than `_POSIX_DELAYTIMER_MAX`, or -1 (ERROR) if *timerid* is invalid.

**ERRNO** EINVAL, ENOSYS

**SEE ALSO** *timerLib*

***timer\_gettime()***

---

***timer\_gettime()***

**NAME** *timer\_gettime()* – get the remaining time before expiration and the reload value (POSIX)

**SYNOPSIS**

```
int timer_gettime
(
 timer_t timerid, /* timer ID */
 struct itimerspec * value /* where to return remaining time */
)
```

**DESCRIPTION** This routine gets the remaining time and reload value of a specified timer. Both values are copied to the *value* structure.

**RETURNS** 0 (OK), or -1 (ERROR) if *timerid* is invalid.

**ERRNO** EINVAL

**SEE ALSO** timerLib

---

***timer\_settime()***

**NAME** *timer\_settime()* – set the time until the next expiration and arm timer (POSIX)

**SYNOPSIS**

```
int timer_settime
(
 timer_t timerid, /* timer ID */
 int flags, /* absolute or relative */
 const struct itimerspec * value, /* time to be set */
 struct itimerspec * ovalue /* previous time set (NULL=no result) */
)
```

**DESCRIPTION** This routine sets the next expiration of the timer, using the *.it\_value* of *value*, thus arming the timer. If the timer is already armed, this call resets the time until the next expiration. If *.it\_value* is zero, the timer is disarmed.

If *flags* is not equal to **TIMER\_ABSTIME**, the interval is relative to the current time, the expiration is set to the difference between the absolute time of *.it\_value* and the current value of the clock associated with *timerid*. If the time has already passed, then the timer expiration notification is made immediately. The task that sets the timer receives the

signal; in other words, the `taskId` is noted. If a timer is set by an ISR, the signal is delivered to the task that created the timer.

The reload value of the timer is set to the value specified by the `.it_interval` field of `value`. When a timer is armed with a nonzero `.it_interval` a periodic timer is set up.

Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer are rounded up to the larger multiple of the resolution.

If `ovalue` is non-NULL, the routine stores a value representing the previous amount of time before the timer would have expired. Or if the timer is disarmed, the routine stores zero, together with the previous timer reload value. The `ovalue` parameter is the same value as that returned by `timer_gettime()` and is subject to the timer resolution.

**WARNING** If `clock_settime()` is called to reset the absolute clock time after a timer has been set with `timer_settime()`, and if `flags` is equal to `TIMER_ABSTIME`, then the timer will behave unpredictably. If you must reset the absolute clock time after setting a timer, do not use `flags` equal to `TIMER_ABSTIME`.

**RETURNS** 0 (OK), or -1 (ERROR) if `timerid` is invalid, the number of nanoseconds specified by `value` is less than 0 or greater than or equal to 1,000,000,000, or the time specified by `value` exceeds the maximum allowed by the timer.

**ERRNO** EINVAL

**SEE ALSO** `timerLib`

---

## *timex()*

**NAME** `timex()` – time a single execution of a function or functions

**SYNOPSIS**

```
void timex
(
 FUNCPTR func, /* function to time (optional) */
 int arg1, /* first of up to 8 args to call func with (optional) */
 int arg2,
 int arg3,
 int arg4,
 int arg5,
 int arg6,
 int arg7,
 int arg8
)
```

***timexClear()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine times a single execution of a specified function with up to eight of the function's arguments. If no function is specified, it times the execution of the current list of functions to be timed, which is created using <i>timexFunc()</i> , <i>timexPre()</i> , and <i>timexPost()</i> . If <i>timex()</i> is executed with a function argument, the entire current list is replaced with the single specified function.<br><br>When execution is complete, <i>timex()</i> displays the execution time. If the execution was so fast relative to the clock rate that the time is meaningless (error > 50%), a warning message is printed instead. In such cases, use <i>timexN()</i> . |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <i>timexLib</i> , <i>timexFunc()</i> , <i>timexPre()</i> , <i>timexPost()</i> , <i>timexN()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

---

***timexClear()***

|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <b>NAME</b>        | <i>timexClear()</i> – clear the list of function calls to be timed |
| <b>SYNOPSIS</b>    | <code>void timexClear (void)</code>                                |
| <b>DESCRIPTION</b> | This routine clears the current list of functions to be timed.     |
| <b>RETURNS</b>     | N/A                                                                |
| <b>SEE ALSO</b>    | <i>timexLib</i>                                                    |

---

***timexFunc()***

|                 |                                                                                                                                                                                                                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <i>timexFunc()</i> – specify functions to be timed                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b> | <pre>void timexFunc (     int    i,      /* function number in list (0..3) */     FUNCPTR func, /* function to be added (NULL if to be deleted) */     int    arg1, /* first of up to 8 args to call function with */     int    arg2,     int    arg3,     int    arg4,     int    arg5,     int    arg6,</pre> |



```

int arg7,
int arg8
)

```

- DESCRIPTION** This routine adds or deletes functions in the list of functions to be timed as a group by calls to *timex()* or *timexN()*. Up to four functions can be included in the list. The argument *i* specifies the function's position in the sequence of execution (0, 1, 2, or 3). A function is deleted by specifying its sequence number *i* and NULL for the function argument *func*.
- RETURNS** N/A
- SEE ALSO** *timexLib*, *timex()*, *timexN()*

---

## *timexHelp()*

- NAME** *timexHelp()* – display synopsis of execution timer facilities
- SYNOPSIS** `void timexHelp (void)`
- DESCRIPTION** This routine displays the following summary of the available execution timer functions:
- |                         |                               |                                               |
|-------------------------|-------------------------------|-----------------------------------------------|
| <code>timexHelp</code>  |                               | Print this list.                              |
| <code>timex</code>      | <code>[func,[args...]]</code> | Time a single execution.                      |
| <code>timexN</code>     | <code>[func,[args...]]</code> | Time repeated executions.                     |
| <code>timexClear</code> |                               | Clear all functions.                          |
| <code>timexFunc</code>  | <code>i,func,[args...]</code> | Add timed function number <i>i</i> (0,1,2,3). |
| <code>timexPre</code>   | <code>i,func,[args...]</code> | Add pre-timing function number <i>i</i> .     |
| <code>timexPost</code>  | <code>i,func,[args...]</code> | Add post-timing function number <i>i</i> .    |
| <code>timexShow</code>  |                               | Show all functions to be called.              |
- Notes:**
- 1) `timexN()` will repeat calls enough times to get timing accuracy to approximately 2%.
  - 2) A single function can be specified with `timex()` and `timexN()`; or, multiple functions can be pre-set with `timexFunc()`.
  - 3) Up to 4 functions can be pre-set with `timexFunc()`, `timexPre()`, and `timexPost()`, i.e., *i* in the range 0 - 3.
  - 4) `timexPre()` and `timexPost()` allow locking/unlocking, or raising/lowering priority before/after timing.
- RETURNS** N/A
- SEE ALSO** *timexLib*

## *timexInit()*

|                    |                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>timexInit()</i> – include the execution timer library                                                                                                                                                                   |
| <b>SYNOPSIS</b>    | <code>void timexInit (void)</code>                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | This null routine is provided so that <b>timexLib</b> can be linked into the system. If the configuration macro <b>INCLUDE_TIMEX</b> is defined, it is called by the root task, <i>usrRoot()</i> , in <i>usrConfig.c</i> . |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                        |
| <b>SEE ALSO</b>    | <b>timexLib</b>                                                                                                                                                                                                            |

---

## *timexN()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>timexN()</i> – time repeated executions of a function or group of functions                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>void timexN (     FUNCPTR func, /* function to time (optional) */     int     arg1, /* first of up to 8 args to call function with */     int     arg2,     int     arg3,     int     arg4,     int     arg5,     int     arg6,     int     arg7,     int     arg8 )</pre>                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | This routine times the execution of the current list of functions to be timed in the same manner as <i>timex()</i> ; however, the list of functions is called a variable number of times until sufficient resolution is achieved to establish the time with an error less than 2%. (Since each iteration of the list may be measured to a resolution of +/- 1 clock tick, repetitive timings decrease this error to 1/N ticks, where N is the number of repetitions.) |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <b>timexLib</b> , <i>timexFunc()</i> , <i>timex()</i>                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

## *timexPost()*

**NAME** *timexPost()* – specify functions to be called after timing

**SYNOPSIS**

```
void timexPost
(
 int i, /* function number in list (0..3) */
 FUNCPTR func, /* function to be added (NULL if to be deleted) */
 int arg1, /* first of up to 8 args to call function with */
 int arg2,
 int arg3,
 int arg4,
 int arg5,
 int arg6,
 int arg7,
 int arg8
)
```

**DESCRIPTION** This routine adds or deletes functions in the list of functions to be called immediately following the timed functions. A maximum of four functions may be included. Up to eight arguments may be passed to each function.

**RETURNS** N/A

**SEE ALSO** *timexLib*

---

## *timexPre()*

**NAME** *timexPre()* – specify functions to be called prior to timing

**SYNOPSIS**

```
void timexPre
(
 int i, /* function number in list (0..3) */
 FUNCPTR func, /* function to be added (NULL if to be deleted) */
 int arg1, /* first of up to 8 args to call function with */
 int arg2,
 int arg3,
 int arg4,
 int arg5,
 int arg6,
```

***timexShow()***

```

 int arg7,
 int arg8
)

```

|                    |                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine adds or deletes functions in the list of functions to be called immediately prior to the timed functions. A maximum of four functions may be included. Up to eight arguments may be passed to each function. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>timexLib</b>                                                                                                                                                                                                           |

---

***timexShow()***

|                    |                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>timexShow()</i> – display the list of function calls to be timed                                                                                                         |
| <b>SYNOPSIS</b>    | <code>void timexShow (void)</code>                                                                                                                                          |
| <b>DESCRIPTION</b> | This routine displays the current list of function calls to be timed. These lists are created by calls to <i>timexPre()</i> , <i>timexFunc()</i> , and <i>timexPost()</i> . |
| <b>RETURNS</b>     | N/A                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <b>timexLib</b> , <i>timexPre()</i> , <i>timexFunc()</i> , <i>timexPost()</i>                                                                                               |

---

***tmpfile()***

|                      |                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>tmpfile()</i> – create a temporary binary file (Unimplemented) (ANSI)                       |
| <b>SYNOPSIS</b>      | <code>FILE * tmpfile (void)</code>                                                             |
| <b>DESCRIPTION</b>   | This routine is not be implemented because VxWorks does not close all open files at task exit. |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                 |
| <b>RETURNS</b>       | NULL                                                                                           |
| <b>SEE ALSO</b>      | <b>ansiStdio</b>                                                                               |

---

## *tmpnam()*

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>tmpnam()</i> – generate a temporary file name (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>      | <pre>char * tmpnam (     char * s /* name buffer */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b>   | <p>This routine generates a string that is a valid file name and not the same as the name of an existing file. It generates a different string each time it is called, up to <code>TMP_MAX</code> times.</p> <p>If the argument is a null pointer, <i>tmpnam()</i> leaves its result in an internal static object and returns a pointer to that object. Subsequent calls to <i>tmpnam()</i> may modify the same object. If the argument is not a null pointer, it is assumed to point to an array of at least <code>L_tmpnam</code> chars; <i>tmpnam()</i> writes its result in that array and returns the argument as its value.</p> |
| <b>INCLUDE FILES</b> | <code>stdio.h</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>RETURNS</b>       | A pointer to the file name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>      | <code>ansiStdio</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

---

## *tolower()*

|                      |                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>tolower()</i> – convert an upper-case letter to its lower-case equivalent (ANSI)                                      |
| <b>SYNOPSIS</b>      | <pre>int tolower (     int c /* character to convert */ )</pre>                                                          |
| <b>DESCRIPTION</b>   | This routine converts an upper-case letter to the corresponding lower-case letter.                                       |
| <b>INCLUDE FILES</b> | <code>ctype.h</code>                                                                                                     |
| <b>RETURNS</b>       | If <i>c</i> is an upper-case letter, it returns the lower-case equivalent; otherwise, it returns the argument unchanged. |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                                                                   |

## ***toupper()***

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>toupper()</i> – convert a lower-case letter to its upper-case equivalent (ANSI)                                      |
| <b>SYNOPSIS</b>      | <pre>int toupper (     int c /* character to convert */ )</pre>                                                         |
| <b>DESCRIPTION</b>   | This routine converts a lower-case letter to the corresponding upper-case letter.                                       |
| <b>INCLUDE FILES</b> | <code>cctype.h</code>                                                                                                   |
| <b>RETURNS</b>       | If <i>c</i> is a lower-case letter, it returns the upper-case equivalent; otherwise, it returns the argument unchanged. |
| <b>SEE ALSO</b>      | <code>ansiCtype</code>                                                                                                  |

---

## ***tr()***

|                    |                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tr()</i> – resume a task                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>void tr (     int taskNameOrId /* task name or task ID */ )</pre>                                                                                               |
| <b>DESCRIPTION</b> | This command resumes the execution of a suspended task. It simply calls <i>taskResume()</i> .                                                                        |
| <b>RETURNS</b>     | N/A                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <code>usrLib</code> , <i>ts()</i> , <i>taskResume()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , <code>windsh</code> , <i>Tornado User's Guide: Shell</i> |

---

## *trunc()*

**NAME** *trunc()* – truncate to integer

**SYNOPSIS**

```
double trunc
(
 double x /* value to truncate */
)
```

**DESCRIPTION** This routine discards the fractional part of a double-precision value *x*.

**INCLUDE FILES** `math.h`

**RETURNS** The integer portion of *x*, represented in double-precision.

**SEE ALSO** `mathALib`

---

## *truncf()*

**NAME** *truncf()* – truncate to integer

**SYNOPSIS**

```
float truncf
(
 float x /* value to truncate */
)
```

**DESCRIPTION** This routine discards the fractional part of a single-precision value *x*.

**INCLUDE FILES** `math.h`

**RETURNS** The integer portion of *x*, represented in single precision.

**SEE ALSO** `mathALib`

**ts()**

---

**ts()**

|                    |                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ts()</i> – suspend a task                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>void ts (     int taskNameOrId /* task name or task ID */ )</pre>                                                                                    |
| <b>DESCRIPTION</b> | This command suspends the execution of a specified task. It simply calls <i>taskSuspend()</i> .                                                           |
| <b>RETURNS</b>     | N/A                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <i>tr()</i> , <i>taskSuspend()</i> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i> |

---

**tsp()**

|                    |                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tsp()</i> – return the contents of register <b>sp</b> (i960)                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>int tsp (     int taskId /* task ID, 0 means default task */ )</pre>                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This command extracts the contents of register <b>sp</b>, the stack pointer, from the TCB of a specified task. If <i>taskId</i> is omitted or 0, the current default task is assumed.</p> <p>Note: The name <i>tsp()</i> is used because <i>sp()</i> (the logical name choice) conflicts with the routine <i>sp()</i> for spawning a task with default parameters.</p> |
| <b>RETURNS</b>     | The contents of the <b>sp</b> register.                                                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <b>dbgArchLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                                                       |



---

**tt()****NAME** `tt()` – display a stack trace of a task**SYNOPSIS**  

```
STATUS tt
(
 int taskNameOrId /* task name or task ID */
)
```

**DESCRIPTION** This routine displays a list of the nested routine calls that the specified task is in. Each routine call and its parameters are shown.

If *taskNameOrId* is not specified or zero, the last task referenced is assumed. The *tt()* routine can only trace the stack of a task other than itself. For instance, when *tt()* is called from the shell, it cannot trace the shell's stack.

**EXAMPLE**  

```
-> tt "logTask"
3ab92 _vxTaskEntry +10 : _logTask (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
ee6e _logTask +12 : _read (5, 3f8a10, 20)
d460 _read +10 : _iosRead (5, 3f8a10, 20)
e234 _iosRead +9c : _pipeRead (3f8a10, 20)
23978 _pipeRead +24 : _semTake (3f8b78)
value = 0 = 0x0
```

This indicates that *logTask()* is currently in *semTake()* (with one parameter) and was called by *pipeRead()* (with three parameters), which was called by *iosRead()* (with three parameters), and so on.

**CAVEAT** In order to do the trace, some assumptions are made. In general, the trace will work for all C language routines and for assembly language routines that start with a LINK instruction. Some C compilers require specific flags to generate the LINK first. Most VxWorks assembly language routines include LINK instructions for this reason. The trace facility may produce inaccurate results or fail completely if the routine is written in a language other than C, the routine's entry point is non-standard, or the task's stack is corrupted. Also, all parameters are assumed to be 32-bit quantities, so structures passed as parameters will be displayed as *long* integers.**RETURNS** OK, or ERROR if the task does not exist.**SEE ALSO** *dbgLib*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

## *ttyDevCreate()*

**NAME** *ttyDevCreate()* – create a VxWorks device for a serial channel

**SYNOPSIS**

```
STATUS ttyDevCreate
(
 char * name, /* name to use for this device */
 SIO_CHAN * pSioChan, /* pointer to core driver structure */
 int rdBufSize, /* read buffer size, in bytes */
 int wrtBufSize /* write buffer size, in bytes */
)
```

**DESCRIPTION** This routine creates a device on a specified serial channel. Each channel to be used should have exactly one device associated with it by calling this routine.

For instance, to create the device `"/tyCo/0"`, with buffer sizes of 512 bytes, the proper call would be:

```
 ttyDevCreate ("/tyCo/0", pSioChan, 512, 512);
```

Where `pSioChan` is the address of the underlying `SIO_CHAN` serial channel descriptor (defined in `sioLib.h`). This routine is typically called by `usrRoot()` in `usrConfig.c`

**RETURNS** OK, or ERROR if the driver is not installed, or the device already exists.

**SEE ALSO** `ttyDrv`

---

## *ttyDrv()*

**NAME** *ttyDrv()* – initialize the tty driver

**SYNOPSIS** STATUS `ttyDrv (void)`

**DESCRIPTION** This routine initializes the tty driver, which is the OS interface to core serial channel(s). Normally, it is called by `usrRoot()` in `usrConfig.c`.

After this routine is called, `ttyDevCreate()` is typically called to bind serial channels to VxWorks devices.

**RETURNS** OK, or ERROR if the driver cannot be installed.

**SEE ALSO** `ttyDrv`

---

## *tyAbortFuncSet()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tyAbortFuncSet()</i> – set the abort function                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>void tyAbortFuncSet (     FUNCPTR func /* routine to call when abort char received */ )</pre>                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine sets the function that will be called when the abort character is received on a tty. There is only one global abort function, used for any tty on which <b>OPT_ABORT</b> is enabled. When the abort character is received from a tty with <b>OPT_ABORT</b> set, the function specified in <i>func</i> will be called, with no parameters, from interrupt level.</p> <p>Setting an abort function of NULL will disable the abort function.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <i>tyLib</i> , <i>tyAbortSet()</i>                                                                                                                                                                                                                                                                                                                                                                                                                            |

---

## *tyAbortSet()*

|                    |                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tyAbortSet()</i> – change the abort character                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>void tyAbortSet (     char ch /* char to be abort */ )</pre>                                                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This routine sets the abort character to <i>ch</i>. The default abort character is CTRL+C.</p> <p>Typing the abort character to any device whose <b>OPT_ABORT</b> option is set will cause the shell task to be killed and restarted. Note that the character set by this routine applies to all devices whose handlers use the standard tty package <b>tyLib</b>.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <i>tyLib</i> , <i>tyAbortFuncSet()</i>                                                                                                                                                                                                                                                                                                                                    |

---

## *tyBackspaceSet()*

**NAME** *tyBackspaceSet()* – change the backspace character

**SYNOPSIS**

```
void tyBackspaceSet
(
 char ch /* char to be backspace */
)
```

**DESCRIPTION** This routine sets the backspace character to *ch*. The default backspace character is CTRL+H.

Typing the backspace character to any device operating in line protocol mode (**OPT\_LINE** set) will cause the previous character typed to be deleted, up to the beginning of the current line. Note that the character set by this routine applies to all devices whose handlers use the standard tty package **tyLib**.

**RETURNS** N/A

**SEE ALSO** **tyLib**

---

## *tyDeleteLineSet()*

**NAME** *tyDeleteLineSet()* – change the line-delete character

**SYNOPSIS**

```
void tyDeleteLineSet
(
 char ch /* char to be line-delete */
)
```

**DESCRIPTION** This routine sets the line-delete character to *ch*. The default line-delete character is CTRL+U.

Typing the delete character to any device operating in line protocol mode (**OPT\_LINE** set) will cause all characters in the current line to be deleted. Note that the character set by this routine applies to all devices whose handlers use the standard tty package **tyLib**.

**RETURNS** N/A

**SEE ALSO** **tyLib**

---

## *tyDevInit()*

**NAME** *tyDevInit()* – initialize the tty device descriptor

**SYNOPSIS**

```
STATUS tyDevInit
(
 TY_DEV_ID pTyDev, /* ptr to tty dev descriptor to init */
 int rdBufSize, /* size of read buffer in bytes */
 int wrtBufSize, /* size of write buffer in bytes */
 FUNCPTR txStartup /* device transmit start-up routine */
)
```

**DESCRIPTION** This routine initializes a tty device descriptor according to the specified parameters. The initialization includes allocating read and write buffers of the specified sizes from the memory pool, and initializing their respective buffer descriptors. The semaphores are initialized and the write semaphore is given to enable writers. Also, the transmitter start-up routine pointer is set to the specified routine. All other fields in the descriptor are zeroed.

This routine should be called only by serial drivers.

**RETURNS** OK, or ERROR if there is not enough memory to allocate data structures.

**SEE ALSO** *tyLib*

---

## *tyEOFSet()*

**NAME** *tyEOFSet()* – change the end-of-file character

**SYNOPSIS**

```
void tyEOFSet
(
 char ch /* char to be EOF */
)
```

**DESCRIPTION** This routine sets the EOF character to *ch*. The default EOF character is CTRL-D.

Typing the EOF character to any device operating in line protocol mode (*OPT\_LINE* set) will cause no character to be entered in the current line, but will cause the current line to be terminated (thus without a newline character). The line is made available to reading tasks. Thus, if the EOF character is the first character input on a line, a line length of zero characters is returned to the reader. This is the standard end-of-file indication on a read

***tyIoctl()***

call. Note that the EOF character set by this routine will apply to all devices whose handlers use the standard tty package **tyLib**.

**RETURNS** N/A

**SEE ALSO** **tyLib**

---

## ***tyIoctl()***

**NAME** *tyIoctl()* – handle device control requests

**SYNOPSIS**

```
STATUS tyIoctl
(
 TY_DEV_ID pTyDev, /* ptr to device to control */
 int request, /* request code */
 int arg /* some argument */
)
```

**DESCRIPTION** This routine handles *ioctl()* requests for tty devices. The I/O control functions for tty devices are described in the manual entry for **tyLib**.

**BUGS** In line protocol mode (**OPT\_LINE** option set), the **FIONREAD** function actually returns the number of characters available plus the number of lines in the buffer. Thus, if five lines consisting of just **NEWLINES** were in the input buffer, the **FIONREAD** function would return the value ten (five characters + five lines).

**RETURNS** OK or ERROR.

**SEE ALSO** **tyLib**

---

## ***tyIRd()***

**NAME** *tyIRd()* – interrupt-level input

**SYNOPSIS**

```
STATUS tyIRd
(
 TY_DEV_ID pTyDev, /* ptr to tty device descriptor */
 char inchar /* character read */
)
```

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | <p>This routine handles interrupt-level character input for tty devices. A device driver calls this routine when it has received a character. This routine adds the character to the ring buffer for the specified device, and gives a semaphore if a task is waiting for it.</p> <p>This routine also handles all the special characters, as specified in the option word for the device, such as X-on, X-off, NEWLINE, or backspace.</p> |
| <b>RETURNS</b>     | OK, or ERROR if the ring buffer is full.                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <b>tyLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                               |

---

## ***tyITx()***

|                    |                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tyITx()</i> – interrupt-level output                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>STATUS tyITx (     TY_DEV_ID pTyDev, /* pointer to tty device descriptor */     char *    pChar   /* where to put character to be output */ )</pre>                                                                                      |
| <b>DESCRIPTION</b> | <p>This routine gets a single character to be output to a device. It looks at the ring buffer for <i>pTyDev</i> and gives the caller the next available character, if there is one. The character to be output is copied to <i>pChar</i>.</p> |
| <b>RETURNS</b>     | OK if there are more characters to send, or ERROR if there are no more characters.                                                                                                                                                            |
| <b>SEE ALSO</b>    | <b>tyLib</b>                                                                                                                                                                                                                                  |

---

## ***tyMonitorTrapSet()***

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <b>NAME</b>     | <i>tyMonitorTrapSet()</i> – change the trap-to-monitor character               |
| <b>SYNOPSIS</b> | <pre>void tyMonitorTrapSet (     char ch /* char to be monitor trap */ )</pre> |

***tyRead()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | <p>This routine sets the trap-to-monitor character to <i>ch</i>. The default trap-to-monitor character is CTRL+X.</p> <p>Typing the trap-to-monitor character to any device whose <b>OPT_MON_TRAP</b> option is set will cause the resident ROM monitor to be entered, if one is present. Once the ROM monitor is entered, the normal multitasking system is halted.</p> <p>Note that the trap-to-monitor character set by this routine will apply to all devices whose handlers use the standard tty package <b>tyLib</b>. Also note that not all systems have a monitor trap available.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>tyLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

---

***tyRead()***

|                    |                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>tyRead()</i> – do a task-level read for a tty device                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>int tyRead (     TY_DEV_ID pTyDev, /* device to read */     char *    buffer, /* buffer to read into */     int      maxbytes /* maximum length of read */ ) </pre>                                                     |
| <b>DESCRIPTION</b> | <p>This routine handles the task-level portion of the tty handler's read function. It reads into the buffer up to <i>maxbytes</i> available bytes.</p> <p>This routine should only be called from serial device drivers.</p> |
| <b>RETURNS</b>     | The number of bytes actually read into the buffer.                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>tyLib</b>                                                                                                                                                                                                                 |



---

## *tyWrite()*

**NAME** *tyWrite()* – do a task-level write for a tty device

**SYNOPSIS**

```
int tyWrite
(
 TY_DEV_ID pTyDev, /* ptr to device structure */
 char * buffer, /* buffer of data to write */
 int nbytes /* number of bytes in buffer */
)
```

**DESCRIPTION** This routine handles the task-level portion of the tty handler's write function.

**RETURNS** The number of bytes actually written to the device.

**SEE ALSO** *tyLib*

---

## *udpShowInit()*

**NAME** *udpShowInit()* – initialize UDP show routines

**SYNOPSIS**

```
void udpShowInit (void)
```

**DESCRIPTION** This routine links the UDP show facility into the VxWorks system. These routines are included automatically if `INCLUDE_NET_SHOW` and `INCLUDE_UDP` are defined in `configAll.h`.

**RETURNS** N/A

**SEE ALSO** *udpShow*

---

## *udpstatShow()*

**NAME** *udpstatShow()* – display statistics for the UDP protocol

**SYNOPSIS**

```
void udpstatShow (void)
```

***ulattach()***

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine displays statistics for the UDP protocol. |
| <b>RETURNS</b>     | N/A                                                    |
| <b>SEE ALSO</b>    | <b>udpShow</b>                                         |

---

***ulattach()***

|                    |                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ulattach()</i> – attach a ULIP interface to a list of network interfaces (VxSim)                                                                         |
| <b>SYNOPSIS</b>    | <pre>STATUS ulattach (     int unit /* ULIP unit number */ )</pre>                                                                                          |
| <b>DESCRIPTION</b> | This routine is called by <i>ulipInit()</i> . It inserts a pointer to the ULIP interface data structure into a linked list of available network interfaces. |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                |
| <b>ERRNO</b>       | S_if_ul_UNIT_ALREADY_INITIALIZED                                                                                                                            |
| <b>SEE ALSO</b>    | <i>if_ulip</i> , <i>VxSim User's Guide</i>                                                                                                                  |

---

***ulipDebugSet()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ulipDebugSet()</i> – Set debug flag in UNIX's ULIP driver                                                                                                                                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>STATUS ulipDebugSet (     int debugFlag )</pre>                                                                                                                                                                                                                                                                                                                                                                        |
| <b>DESCRIPTION</b> | This function uses an ioctl call to UNIX's (Solaris's) ULIP driver to set that driver's debugging flag to the value in debugFlag. Because there is no simple way for the caller to ascertain the unit number of the interface in use, all unit numbers are looped over and each receives the ioctl. Possible values for the debug flag are discussed above in this file, although all the levels have not been implemented. |

This is not the right place to put this function (user callable routines would be more appropriately placed in **simLib.h**). Because of the requirement to use both Sun structures (to bundle ioctl data) and VxWorks structures (**ul\_softc**), and given the same requirements when calling the **FIOSETUSED** ioctl, this seems the best place to put it.

**RETURNS** OK or ERROR if the ioctl fails

**SEE ALSO** **if\_ulip**

---

## *ulipDelete()*

**NAME** *ulipDelete()* – delete a ULIP interface (VxSim)

**SYNOPSIS**

```
STATUS ulipDelete
(
 int unit /* ULIP unit number */
)
```

**DESCRIPTION** This routine detaches the ULIP unit and frees up system resources taken up by this ULIP interface.

**RETURNS** OK, or ERROR if the unit number is invalid or the interface is uninitialized.

**ERRNO** **S\_if\_ul\_INVALID\_UNIT\_NUMBER**, **S\_if\_ul\_UNIT\_UNINITIALIZED**

**SEE ALSO** **if\_ulip**, *VxSim User's Guide*

---

## *ulipInit()*

**NAME** *ulipInit()* – initialize the ULIP interface (VxSim)

**SYNOPSIS**

```
STATUS ulipInit
(
 int unit, /* ULIP unit number (0 - NULIP-1) */
 char * myAddr, /* IP address of the interface */
 char * peerAddr, /* IP address of the remote peer interface */
 int procnum /* processor number to map to ULIP interface */
)
```

|                    |                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine initializes the ULIP interface and sets the Internet address as a function of the processor number. |
| <b>RETURNS</b>     | OK, or ERROR if the device cannot be opened or there is insufficient memory.                                     |
| <b>ERRNO</b>       | S_if_ul_INVALID_UNIT_NUMBER                                                                                      |
| <b>SEE ALSO</b>    | <i>if_ulip</i> , <i>VxSim User's Guide</i>                                                                       |

---

## ***ulStartOutput()***

|                 |                                                                            |
|-----------------|----------------------------------------------------------------------------|
| <b>NAME</b>     | <i>ulStartOutput()</i> – push packets onto "interface"                     |
| <b>SYNOPSIS</b> | <pre>#ifdef BSD43_DRIVER LOCAL STATUS ulStartOutput (     int unit )</pre> |
| <b>SEE ALSO</b> | <i>if_ulip</i>                                                             |

---

## ***ultraAddrFilterSet()***

|                    |                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ultraAddrFilterSet()</i> – set the address filter for multicast addresses                                                                                              |
| <b>SYNOPSIS</b>    | <pre>void ultraAddrFilterSet (     ULTRA_DEVICE * pDrvCtrl /* device pointer */ )</pre>                                                                                   |
| <b>DESCRIPTION</b> | This routine goes through all of the multicast addresses on the list of addresses (added with the <i>ultraMCastAdd()</i> routine) and sets the device's filter correctly. |
| <b>RETURNS</b>     | N/A.                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <i>ultraEnd</i>                                                                                                                                                           |

---

## *ultraattach()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ultraattach()</i> – publish <b>ultra</b> interface and initialize device                                                                                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS ultraattach (     int unit,      /* unit number */     int ioAddr,   /* address of ultra's shared memory */     int ivec,     /* interrupt vector to connect to */     int ilevel,   /* interrupt level */     int memAddr,  /* address of ultra's shared memory */     int memSize,  /* size of ultra's shared memory */     int config    /* 0: RJ45 + AUI(Thick) 1: RJ45 + BNC(Thin) */ )</pre> |
| <b>DESCRIPTION</b> | This routine attaches an <b>ultra</b> Ethernet interface to the network if the device exists. It makes the interface available by filling in the network interface record. The system will initialize the interface when it is ready to accept packets.                                                                                                                                                        |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <i>if_ultra</i> , <i>ifLib</i> , <i>netShow</i>                                                                                                                                                                                                                                                                                                                                                                |

---

## *ultraLoad()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>ultraLoad()</i> – initialize the driver and device                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>END_OBJ* ultraLoad (     char * initString /* String to be parsed by the driver. */ )</pre>                                                                                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b> | <p>This routine initializes the driver and the device to the operational state. All of the device-specific parameters are passed in <i>initString</i>, which expects a string of the following format:</p> <pre>unit:ioAddr:memAddr:vecNum:intLvl:config:offset"</pre> <p>This routine can be called in two modes. If it is called with an empty but allocated string, it places the name of this device (that is, "ultra") into the <i>initString</i> and returns 0.</p> |

***ultraMemInit()***

If the string is allocated and not empty, the routine attempts to load the driver using the values specified in the string.

**RETURNS** An END object pointer, or NULL on error, or 0 and the name of the device if the *initString* was NULL.

**SEE ALSO** **ultraEnd**

***ultraMemInit()***

**NAME** *ultraMemInit()* – initialize memory for the chip

**SYNOPSIS**

```
STATUS ultraMemInit
(
 ULTRA_DEVICE * pDrvCtrl, /* device to be initialized */
 int clNum /* number of clusters to allocate */
)
```

**DESCRIPTION** Using data in the control structure, setup and initialize the memory areas needed. If the memory address is not already specified, then allocate cache safe memory.

**RETURNS** OK or ERROR.

**SEE ALSO** **ultraEnd**

***ultraParse()***

**NAME** *ultraParse()* – parse the init string

**SYNOPSIS**

```
STATUS ultraParse
(
 ULTRA_DEVICE * pDrvCtrl, /* device pointer */
 char * initString /* information string */
)
```

**DESCRIPTION** Parse the input string. Fill in values in the driver control structure. The initialization string format is: *unit:ioAddr:memAddr:vecNum:intLvl:config:offset*

*unit*  
Device unit number, a small integer.

*ioAddr*  
I/O address

*memAddr*  
Memory address, assumed to be 16k bytes in length.

*vecNum*  
Interrupt vector number (used with *sysIntConnect()*).

*intLvl*  
Interrupt level.

*config*  
Ultra config (0: RJ45 + AUI(Thick) 1: RJ45 + BNC(Thin)).

*offset*  
Memory offset for alignment.

**RETURNS** OK, or ERROR if any arguments are invalid.

**SEE ALSO** [ultraEnd](#)

---

## ***ultraPut()***

**NAME** *ultraPut()* – copy a packet to the interface.

**SYNOPSIS**

```
#ifdef BSD43_DRIVER LOCAL void ultraPut
(
 int unit /* device unit number */
)
```

**DESCRIPTION** Copy from mbuf chain to transmitter buffer in shared memory.

**RETURNS** N/A

**SEE ALSO** [if\\_ultra](#)

**ultraShow()**

---

**ultraShow()****NAME** *ultraShow()* – display statistics for the **ultra** network interface**SYNOPSIS**  

```
void ultraShow
(
 int unit, /* interface unit */
 BOOL zap /* zero totals */
)
```

**DESCRIPTION** This routine displays statistics about the **elc** Ethernet network interface. It has two parameters:

*unit*  
 interface unit; should be 0.

*zap*  
 if 1, all collected statistics are cleared to zero.

**RETURNS** N/A**SEE ALSO** **if\_ultra**

---

**ungetc()****NAME** *ungetc()* – push a character back into an input stream (ANSI)**SYNOPSIS**  

```
int ungetc
(
 int c, /* character to push */
 FILE * fp /* input stream */
)
```

**DESCRIPTION** This routine pushes a character *c* (converted to an **unsigned char**) back into the specified input stream. The pushed-back characters will be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call on the stream to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

One character of push-back is guaranteed. If *ungetc()* is called too many times on the same stream without an intervening read or file positioning operation, the operation may fail.



If the value of *c* equals EOF, the operation fails and the input stream is unchanged.

A successful call to *ungetc()* clears the end-of-file indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back characters is the same as it was before the character were pushed back. For a text stream, the value of its file position indicator after a successful call to *ungetc()* is unspecified until all pushed-back characters are read or discarded. For a binary stream, the file position indicator is decremented by each successful call to *ungetc()*; if its value was zero before a call, it is indeterminate after the call.

**INCLUDE** `stdio.h`

**RETURNS** The pushed-back character after conversion, or EOF if the operation fails.

**SEE ALSO** `ansiStdio`, `getc()`, `fgetc()`

---

## *unixDevInit()*

**NAME** *unixDevInit()* – initialize a UNIX\_DUSART

**SYNOPSIS**

```
void unixDevInit
(
 UNIX_CHAN * pChan
)
```

**DESCRIPTION** This routine initializes the driver function pointers and then resets to a quiescent state. The BSP must have already opened all the file descriptors in the structure before passing it to this routine.

**RETURNS** N/A

**SEE ALSO** `unixSio`

## **unixDevInit2()**

**NAME** *unixDevInit2()* – enable interrupts

**SYNOPSIS**

```
void unixDevInit2
(
 UNIX_CHAN * pChan
)
```

**RETURNS** N/A

**SEE ALSO** *unixSio*

---

## **unixDiskDevCreate()**

**NAME** *unixDiskDevCreate()* – create a UNIX disk device

**SYNOPSIS**

```
BLK_DEV *unixDiskDevCreate
(
 char * unixFile, /* name of the UNIX file */
 int bytesPerBlk, /* number of bytes per block */
 int blksPerTrack, /* number of blocks per track */
 int nBlocks /* number of blocks on this device */
)
```

**DESCRIPTION** This routine creates a UNIX disk device.

The *unixFile* parameter specifies the name of the UNIX file to use for the disk device.

The *bytesPerBlk* parameter specifies the size of each logical block on the disk. If *bytesPerBlk* is zero, 512 is the default.

The *blksPerTrack* parameter specifies the number of blocks on each logical track of the disk. If *blksPerTrack* is zero, the count of blocks per track is set to *nBlocks* (i.e., the disk is defined as having only one track).

The *nBlocks* parameter specifies the size of the disk, in blocks. If *nBlocks* is zero, a default size is used. The default is calculated as the size of the UNIX disk divided by the number of bytes per block.

This routine is only applicable to VxSim for Solaris and VxSim for HP.

**RETURNS** A pointer to block device (BLK\_DEV) structure, or NULL, if unable to open the UNIX disk.

SEE ALSO `unixDrv`

---

## *unixDiskInit()*

**NAME** `unixDiskInit()` – initialize a dosFs disk on top of UNIX

**SYNOPSIS**

```
void unixDiskInit
(
 char * unixFile, /* UNIX file name */
 char * volName, /* dosFs name */
 int diskSize /* number of bytes */
)
```

**DESCRIPTION** This routine provides some convenience for a user wanting to create a UNIX disk-based dosFs file system under VxWorks. The user only specifies the UNIX file to use, the dosFs volume name, and the size of the volume in bytes, if the UNIX file needs to be created.

This routine is only applicable to VxSim for Solaris and VxSim for HP.

**RETURNS** N/A

**SEE ALSO** `unixDrv`

---

## *unixDrv()*

**NAME** `unixDrv()` – install UNIX disk driver

**SYNOPSIS** `STATUS unixDrv (void)`

**DESCRIPTION** Used in `usrConfig.c` to cause the UNIX disk driver to be linked in when building VxWorks. Otherwise, it is not necessary to call this routine before using the UNIX disk driver.

This routine is only applicable to VxSim for Solaris and VxSim for HP.

**RETURNS** OK (always).

**SEE ALSO** `unixDrv`

***unixIntRcv()***

---

***unixIntRcv()***

|                 |                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <i>unixIntRcv()</i> – handle a channel’s receive-character interrupt.                       |
| <b>SYNOPSIS</b> | <pre>void unixIntRcv (     UNIX_CHAN * pChan /* channel generating the interrupt */ )</pre> |
| <b>RETURNS</b>  | N/A                                                                                         |
| <b>SEE ALSO</b> | <i>unixSio</i>                                                                              |

---

***unld()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>unld()</i> – unload an object module by specifying a file name or module ID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS unld (     void * nameOrId, /* name or ID of the object module file */     int options )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>This routine unloads the specified object module from the system. The module can be specified by name or by module ID. For a.out and ECOFF format modules, unloading does the following:</p> <ol style="list-style-type: none"> <li>(1) It frees the space allocated for text, data, and BSS segments, unless <i>loadModuleAt()</i> was called with specific addresses, in which case the user is responsible for freeing the space.</li> <li>(2) It removes all symbols associated with the object module from the system symbol table.</li> <li>(3) It removes the module descriptor from the module list.</li> </ol> <p>For other modules of other formats, unloading has similar effects.</p> <p>Before any modules are unloaded, all breakpoints in the system are deleted. If you need to keep breakpoints, set the options parameter to <b>UNLD_KEEP_BREAKPOINTS</b>. No breakpoints can be set in code that is unloaded.</p> |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

SEE ALSO *unldLib*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

---

## *unldByGroup()*

**NAME** *unldByGroup()* – unload an object module by specifying a group number

**SYNOPSIS**

```
STATUS unldByGroup
(
 UINT16 group, /* group number to unload */
 int options /* options, currently unused */
)
```

**DESCRIPTION** This routine unloads an object module that has a group number matching *group*. See the manual entries for *unld()* or *unldLib* for more information on module unloading.

**RETURNS** OK or ERROR.

**SEE ALSO** *unldLib*, *unld()*

---

## *unldByModuleId()*

**NAME** *unldByModuleId()* – unload an object module by specifying a module ID

**SYNOPSIS**

```
STATUS unldByModuleId
(
 MODULE_ID moduleId, /* module ID to unload */
 int options
)
```

**DESCRIPTION** This routine unloads an object module that has a module ID matching *moduleId*. See the manual entries for *unld()* or *unldLib* for more information on module unloading.

**RETURNS** OK or ERROR.

**SEE ALSO** *unldLib*, *unld()*

---

## ***unldByNameAndPath()***

**NAME** *unldByNameAndPath()* – unload an object module by specifying a name and path

**SYNOPSIS**

```
STATUS unldByNameAndPath
(
 char * name, /* name of the object module to unload */
 char * path, /* path to the object module to unload */
 int options /* options, currently unused */
)
```

**DESCRIPTION** This routine unloads an object module specified by *name* and *path*. See the manual entries for *unld()* or **unldLib** for more information on module unloading.

**RETURNS** OK or ERROR.

**SEE ALSO** **unldLib**, *unld()*

---

## ***unlink()***

**NAME** *unlink()* – delete a file (POSIX)

**SYNOPSIS**

```
STATUS unlink
(
 char * name /* name of the file to remove */
)
```

**DESCRIPTION** This routine deletes a specified file. It performs the same function as *remove()* and is provided for POSIX compatibility.

**RETURNS** OK if there is no delete routine for the device or the driver returns OK; ERROR if there is no such device or the driver returns ERROR.

**SEE ALSO** **ioLib**, *remove()*

---

## *usrAtaConfig()*

- NAME** *usrAtaConfig()* – mount a DOS file system from an ATA hard disk
- SYNOPSIS**
- ```
STATUS usrAtaConfig
(
    int    ctrl,    /* 0: primary address, 1: secondary address */
    int    drive,  /* drive number of hard disk (0 or 1) */
    char * fileName /* mount point */
)
```
- DESCRIPTION** This routine mounts a DOS file system from an ATA hard disk. Parameters:
- drive*
the drive number of the hard disk; 0 is **C:** and 1 is **D:**.
- fileName*
the mount point, for example, */ata0/*.
- NOTE** Because VxWorks does not support partitioning, hard disks formatted and initialized on VxWorks are not compatible with DOS machines. This routine does not refuse to mount a hard disk that was initialized on VxWorks. The hard disk is assumed to have only one partition with a partition record in sector 0.
- RETURNS** OK or ERROR.
- SEE ALSO** *src/config/usrAta.c*, *VxWorks Programmer's Guide: I/O System, Local File Systems, Intel i386/i486/Pentium*

usrAtaPartition()

- NAME** *usrAtaPartition()* – get an offset to the first partition of the drive
- SYNOPSIS**
- ```
int usrAtaPartition
(
 int ctrl, /* 0: primary address, 1: secondary address */
 int drive, /* drive number of hard disk (0 or 1) */
 DOS_PART_TBL * pPart /* pointer to the partition table */
)
```

***usrClock()***

|                    |                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine gets an offset to the first partition of the drive. The value of offset is passed to the macro <code>ATA_SWAP</code> for endian adjustment. For the <i>drive</i> parameter, 0 is <b>C:</b> and 1 is <b>D:</b> . |
| <b>RETURNS</b>     | The offset to the partition                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <code>usrAta</code>                                                                                                                                                                                                          |

---

## ***usrClock()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>usrClock()</code> – user-defined system clock interrupt routine                                                                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <code>void usrClock ()</code>                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine is called at interrupt level on each clock interrupt. It is installed by <code>usrRoot()</code> with a <code>sysClkConnect()</code> call. It calls all the other packages that need to know about clock ticks, including the kernel itself.</p> <p>If the application needs anything to happen at the system clock interrupt level, it can be added to this routine.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <code>usrConfig</code>                                                                                                                                                                                                                                                                                                                                                                   |

---

## ***usrFdConfig()***

|                    |                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>usrFdConfig()</code> – mount a DOS file system from a floppy disk                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>STATUS usrFdConfig (     int    drive,    /* drive number of floppy disk (0 - 3) */     int    type,     /* type of floppy disk */     char * fileName /* mount point */ ) </pre> |
| <b>DESCRIPTION</b> | <p>This routine mounts a DOS file system from a floppy disk device.</p> <p>The <i>drive</i> parameter is the drive number of the floppy disk; valid values are 0 to 3.</p>             |



The *type* parameter specifies the type of diskette, which is described in the structure table **fdTypes[]** in **sysLib.c**. *type* is an index to the table. Currently the table contains two diskette types:

- A *type* of 0 indicates the first entry in the table (3.5" 2HD, 1.44MB);
- A *type* of 1 indicates the second entry in the table (5.25" 2HD, 1.2MB).

The *fileName* parameter is the mount point, e.g., */fd0/*.

|                 |                                                                                                                                                                                                                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NOTE</b>     | Do not attempt to unmount a volume that was mounted with <i>usrFdConfig()</i> using <i>dosFsVolUnmount()</i> . <i>usrFdConfig()</i> does not return the <b>DOS_VOL_CONFIG</b> structure required by <i>dosFsVolUnmount()</i> . Instead use <i>ioctl()</i> with <b>FIOUNMOUNT</b> which accesses the volume information via the file descriptor. |
| <b>RETURNS</b>  | OK or ERROR.                                                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b> | <b>usrFd</b> , <i>VxWorks Programmer's Guide: I/O System, Local File Systems, Intel i386/i486 Appendix</i>                                                                                                                                                                                                                                      |

---

## *usrIdeConfig()*

|                    |                                                                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>usrIdeConfig()</i> – mount a DOS file system from an IDE hard disk                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS usrIdeConfig (     int    drive,    /* drive number of hard disk (0 or 1) */     char * fileName /* mount point */ )</pre>                                                                                                                                                                           |
| <b>DESCRIPTION</b> | This routine mounts a DOS file system from an IDE hard disk.<br>The <i>drive</i> parameter is the drive number of the hard disk; 0 is <b>C:</b> and 1 is <b>D:</b> .<br>The <i>fileName</i> parameter is the mount point, e.g., <i>/ide0/</i> .                                                                  |
| <b>NOTE</b>        | Because VxWorks does not support partitioning, hard disks formatted and initialized on VxWorks are not compatible with DOS machines. This routine does not refuse to mount a hard disk that was initialized on VxWorks. The hard disk is assumed to have only one partition with a partition record in sector 0. |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <b>usrIde</b> , <i>VxWorks Programmer's Guide: I/O System, Local File Systems, Intel i386/i486 Appendix</i>                                                                                                                                                                                                      |

## *usrInit()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>usrInit()</i> – user-defined system initialization routine                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>void usrInit (     int startType )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b> | <p>This is the first C code executed after the system boots. This routine is called by the assembly language start-up routine <i>sysInit()</i> which is in the <b>sysALib</b> module of the target-specific directory. It is called with interrupts locked out. The kernel is not multitasking at this point.</p> <p>This routine starts by clearing BSS; thus all variables are initialized to 0, as per the C specification. It then initializes the hardware by calling <i>sysHwInit()</i>, sets up the interrupt/exception vectors, and starts kernel multitasking with <i>usrRoot()</i> as the root task.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | <i>usrConfig</i> , <i>kernelLib</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

---

## *usrRoot()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>usrRoot()</i> – the root task                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>void usrRoot (     char *   pMemPoolStart, /* start of system memory partition */     unsigned memPoolSize   /* initial size of mem pool */ )</pre>                                                                                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This is the first task to run under the multitasking kernel. It performs all final initialization and then starts other tasks.</p> <p>It initializes the I/O system, installs drivers, creates devices, and sets up the network, etc., as necessary for a particular configuration. It may also create and load the system symbol table, if one is to be included. It may then load and spawn additional tasks as needed. In the default configuration, it simply initializes the VxWorks shell.</p> |

**RETURNS** N/A

**SEE ALSO** `usrConfig`

---

## *usrScsiConfig()*

**NAME** *usrScsiConfig()* – configure SCSI peripherals

**SYNOPSIS** `STATUS usrScsiConfig (void)`

**DESCRIPTION** This code configures the SCSI disks and other peripherals on a SCSI controller chain.

The macro `SCSI_AUTO_CONFIG` will include code to scan all possible device/lun id's and to configure a `scsiPhysDev` structure for each device found. Of course this doesn't include final configuration for disk partitions, floppy configuration parameters, or tape system setup. All of these actions must be performed by user code, either through *sysScsiConfig()*, the startup script, or by the application program.

The user may customize this code on a per BSP basis using the `SYS_SCSI_CONFIG` macro. If defined, then this routine will call the routine *sysScsiConfig()*. That routine is to be provided by the BSP, either in `sysLib.c` or `sysScsi.c`. If `SYS_SCSI_CONFIG` is not defined, then *sysScsiConfig()* will not be called as part of this routine.

An example *sysScsiConfig()* routine can be found in `target/src/config/usrScsi.c`. The example code contains sample configurations for a hard disk, a floppy disk and a tape unit.

**RETURNS** OK or ERROR.

**SEE ALSO** *usrScsi*, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## *usrSmObjInit()*

**NAME** *usrSmObjInit()* – initialize shared memory objects

**SYNOPSIS** `STATUS usrSmObjInit`  
`(`  
`char * bootString /* boot parameter string */`  
`)`

***uswab()***

- DESCRIPTION** This routine initializes the shared memory objects facility. It sets up the shared memory objects facility if called from processor 0. Then it initializes a shared memory descriptor and calls *smObjAttach()* to attach this CPU to the shared memory object facility.
- When the shared memory pool resides on the local CPU dual ported memory, **SM\_OBJ\_MEM\_ADRS** must be set to **NONE** in **configAll.h** and the shared memory objects pool is allocated from the VxWorks system pool.
- NOTE** The shared memory objects library requires information from fields in the VxWorks boot line. The functions are contained in the **usrNetwork.c** file. If no network services are included, **usrNetwork.c** is not included and the shared memory initialization fails. To avoid this problem, either add **INCLUDE\_NETWORK** to **configAll.h** or extract the bootline cracking routines from **usrNetwork.c** and include them elsewhere.
- RETURNS** OK, or ERROR if unsuccessful.
- SEE ALSO** **usrSmObj**

---

***uswab()***

- NAME** *uswab()* – swap bytes with buffers that are not necessarily aligned
- SYNOPSIS**
- ```
void uswab
(
    char * source,      /* pointer to source buffer */
    char * destination, /* pointer to destination buffer */
    int  nbytes        /* number of bytes to exchange */
)
```
- DESCRIPTION** This routine gets the specified number of bytes from *source*, exchanges the adjacent even and odd bytes, and puts them in *destination*.
- NOTE** Due to speed considerations, this routine should only be used when absolutely necessary. Use *swab()* for aligned swaps.
- It is an error for *nbytes* to be odd.
- RETURNS** N/A
- SEE ALSO** **bLib**, *swab()*

utime()

NAME *utime()* – update time on a file

SYNOPSIS

```
int utime
(
    char *          file,
    struct utimbuf * newTimes
)
```

RETURNS OK or ERROR.

SEE ALSO *dirLib*, *stat()*, *fstat()*, *ls()*

va_arg()

NAME *va_arg()* – expand to an expression having the type and value of the call's next argument

SYNOPSIS

```
void va_arg
(
)
```

DESCRIPTION Each invocation of this macro modifies an object of type **va_list** (*ap*) so that the values of successive arguments are returned in turn. The parameter *type* is a type name specified such that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a * to *type*. If there is no actual next argument, or if *type* is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined.

RETURNS The first invocation of *va_arg()* after *va_start()* returns the value of the argument after that specified by *parmN* (the rightmost parameter). Successive invocations return the value of the remaining arguments in succession.

SEE ALSO *ansiStdarg*

va_end()

va_end()

| | |
|--------------------|--|
| NAME | <i>va_end()</i> – facilitate a normal return from a routine using a va_list object |
| SYNOPSIS | <pre>void va_end ()</pre> |
| DESCRIPTION | <p>This macro facilitates a normal return from the function whose variable argument list was referred to by the expansion of <i>va_start()</i> that initialized the va_list object.</p> <p><i>va_end()</i> may modify the va_list object so that it is no longer usable (without an intervening invocation of <i>va_start()</i>). If there is no corresponding invocation of the <i>va_start()</i> macro, or if the <i>va_end()</i> macro is not invoked before the return, the behavior is undefined.</p> |
| RETURNS | N/A |
| SEE ALSO | ansiStdarg |

va_start()

| | |
|--------------------|--|
| NAME | <i>va_start()</i> – initialize a va_list object for use by <i>va_arg()</i> and <i>va_end()</i> |
| SYNOPSIS | <pre>void va_start ()</pre> |
| DESCRIPTION | <p>This macro initializes an object of type va_list (<i>ap</i>) for subsequent use by <i>va_arg()</i> and <i>va_end()</i>. The parameter <i>parmN</i> is the identifier of the rightmost parameter in the variable parameter list in the function definition (the one just before the <i>, ...</i>). If <i>parmN</i> is declared with the register storage class with a function or array type, or with a type that is not compatible with the type that results after application of the default argument promotions, the behavior is undefined.</p> |
| RETURNS | N/A |
| SEE ALSO | ansiStdarg |

valloc()

| | |
|--------------------|---|
| NAME | <i>valloc()</i> – allocate memory on a page boundary |
| SYNOPSIS | <pre>void * valloc (unsigned size /* number of bytes to allocate */)</pre> |
| DESCRIPTION | This routine allocates a buffer of <i>size</i> bytes from the system memory partition. Additionally, it insures that the allocated buffer begins on a page boundary. Page sizes are architecture-dependent. |
| RETURNS | A pointer to the newly allocated block, or NULL if the buffer could not be allocated or the memory management unit (MMU) support library has not been initialized. |
| ERRNO | S_memLib_PAGE_SIZE_UNAVAILABLE |
| SEE ALSO | memLib |

version()

| | |
|--------------------|--|
| NAME | <i>version()</i> – print VxWorks version information |
| SYNOPSIS | <pre>void version (void)</pre> |
| DESCRIPTION | This command prints the VxWorks version number, the date this copy of VxWorks was made, and other pertinent information. |
| EXAMPLE | <pre>-> version VxWorks (for Mizar 7170) version 5.1 Kernel: WIND version 2.1. Made on Tue Jul 27 20:26:23 CDT 1997. Boot line: enp(0,0)host:/usr/wpwr/target/config/mz7170/vxWorks e=90.0.0.50 h=90.0.0.4 u=target</pre> |
| RETURNS | N/A |
| SEE ALSO | usrLib , <i>VxWorks Programmer's Guide: Target Shell</i> , windsh , <i>Tornado User's Guide: Shell</i> |

vfdprintf()

NAME *vfdprintf()* – write a string formatted with a variable argument list to a file descriptor

SYNOPSIS

```
int vfdprintf
(
    int          fd,      /* file descriptor to print to */
    const char * fmt,    /* format string for print */
    va_list      vaList /* optional arguments to format */
)
```

DESCRIPTION This routine prints a string formatted with a variable argument list to a specified file descriptor. It is identical to *fdprintf()*, except that it takes the variable arguments to be formatted as a list *vaList* of type **va_list** rather than as in-line arguments.

RETURNS The number of characters output, or ERROR if there is an error during output.

SEE ALSO *fioLib*, *fdprintf()*

vfprintf()

NAME *vfprintf()* – write a formatted string to a stream (ANSI)

SYNOPSIS

```
int vfprintf
(
    FILE *      fp,      /* stream to write to */
    const char * fmt,    /* format string */
    va_list     vaList /* arguments to format string */
)
```

DESCRIPTION This routine is equivalent to *fprintf()*, except that it takes the variable arguments to be formatted from a list *vaList* of type **va_list** rather than from in-line arguments.

INCLUDE FILES *stdio.h*

RETURNS The number of characters written, or a negative value if an output error occurs.

SEE ALSO *ansiStdio*, *fprintf()*

vmBaseGlobalMapInit()

NAME *vmBaseGlobalMapInit()* – initialize global mapping

SYNOPSIS

```

VM_CONTEXT_ID vmBaseGlobalMapInit
(
    PHYS_MEM_DESC * pMemDescArray,      /* pointer to array of mem descs */
    int             numDescArrayElements, /* no. of elements */
                                           /* in pMemDescArray */
    BOOL            enable                /* enable virtual memory */
)
    
```

DESCRIPTION This routine creates and installs a virtual memory context with mappings defined for each contiguous memory segment defined in *pMemDescArray*. In the standard VxWorks configuration, an instance of **PHYS_MEM_DESC** (called **sysPhysMemDesc**) is defined in **sysLib.c**; the variable is passed to *vmBaseGlobalMapInit()* by the system configuration mechanism.

The physical memory descriptor also contains state information used to initialize the state information in the MMU's translation table for that memory segment. The following state bits may be or'ed together:

| | | |
|---------------------------|-------------------------------|--------------------------|
| VM_STATE_VALID | VM_STATE_VALID_NOT | valid/invalid |
| VM_STATE_WRITABLE | VM_STATE_WRITABLE_NOT | writable/write-protected |
| VM_STATE_CACHEABLE | VM_STATE_CACHEABLE_NOT | cacheable/not-cacheable |

Additionally, mask bits are or'ed together in the **initialStateMask** structure element to describe which state bits are being specified in the **initialState** structure element:

```

VM_STATE_MASK_VALID
VM_STATE_MASK_WRITABLE
VM_STATE_MASK_CACHEABLE
    
```

If *enable* is TRUE, the MMU is enabled upon return.

RETURNS A pointer to a newly created virtual memory context, or NULL if memory cannot be mapped.

SEE ALSO *vmBaseLibInit()*

vmBaseLibInit()

| | |
|--------------------|---|
| NAME | <i>vmBaseLibInit()</i> – initialize base virtual memory support |
| SYNOPSIS | <pre>STATUS vmBaseLibInit (int pageSize /* size of page */)</pre> |
| DESCRIPTION | This routine initializes the virtual memory context class and module-specific data structures. It is called only once during system initialization, and should be followed with a call to <i>vmBaseGlobalMapInit()</i> , which initializes and enables the MMU. |
| RETURNS | OK. |
| SEE ALSO | <i>vmBaseLib</i> , <i>vmBaseGlobalMapInit()</i> |

vmBasePageSizeGet()

| | |
|--------------------|--|
| NAME | <i>vmBasePageSizeGet()</i> – return the page size |
| SYNOPSIS | <pre>int vmBasePageSizeGet (void)</pre> |
| DESCRIPTION | This routine returns the architecture-dependent page size. This routine is callable from interrupt level. |
| RETURNS | The page size of the current architecture. |
| SEE ALSO | <i>vmBaseLib</i> |

vmBaseStateSet()

NAME *vmBaseStateSet()* – change the state of a block of virtual memory

SYNOPSIS

```
STATUS vmBaseStateSet
(
    VM_CONTEXT_ID context, /* context - NULL == currentContext */
    void *        pVirtual, /* virtual address to modify state of */
    int          len, /* len of virtual space to modify state of */
    UINT         stateMask, /* state mask */
    UINT         state /* state */
)
```

DESCRIPTION This routine changes the state of a block of virtual memory. Each page of virtual memory has at least three elements of state information: validity, writability, and cacheability. Specific architectures may define additional state information; see **vmLib.h** for additional architecture-specific states. Memory accesses to a page marked as invalid will result in an exception. Pages may be invalidated to prevent them from being corrupted by invalid references. Pages may be defined as read-only or writable, depending on the state of the writable bits. Memory accesses to pages marked as not-cacheable will always result in a memory cycle, bypassing the cache. This is useful for multiprocessing, multiple bus masters, and hardware control registers.

The following states are provided and may be or'ed together in the state parameter:

| | | |
|--------------------|------------------------|--------------------------|
| VM_STATE_VALID | VM_STATE_VALID_NOT | valid/invalid |
| VM_STATE_WRITABLE | VM_STATE_WRITABLE_NOT | writable/write-protected |
| VM_STATE_CACHEABLE | VM_STATE_CACHEABLE_NOT | cacheable/not-cacheable |

Additionally, the following masks are provided so that only specific states may be set. These may be or'ed together in the **stateMask** parameter.

```
VM_STATE_MASK_VALID
VM_STATE_MASK_WRITABLE
VM_STATE_MASK_CACHEABLE
```

If *context* is specified as NULL, the current context is used.

This routine is callable from interrupt level.

RETURNS OK, or ERROR if the validation fails, *pVirtual* is not on a page boundary, *len* is not a multiple of the page size, or the architecture-dependent state set fails for the specified virtual address.

ERRNO S_vmLib_NOT_PAGE_ALIGNED, S_vmLib_BAD_STATE_PARAM, S_vmLib_BAD_MASK_PARAM

SEE ALSO **vmBaseLib**

vmContextCreate()

| | |
|---------------------|---|
| NAME | <i>vmContextCreate()</i> – create a new virtual memory context (VxVMI Opt.) |
| SYNOPSIS | <code>VM_CONTEXT_ID vmContextCreate (void)</code> |
| DESCRIPTION | <p>This routine creates a new virtual memory context. The newly created context does not become the current context until explicitly installed by a call to <i>vmCurrentSet()</i>. Modifications to the context state (mappings, state changes, etc.) may be performed on any virtual memory context, even if it is not the current context.</p> <p>This routine should not be called from interrupt level.</p> |
| AVAILABILITY | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI. |
| RETURNS | A pointer to a new virtual memory context, or NULL if the allocation or initialization fails. |
| SEE ALSO | <code>vmLib</code> |

vmContextDelete()

| | |
|---------------------|--|
| NAME | <i>vmContextDelete()</i> – delete a virtual memory context (VxVMI Opt.) |
| SYNOPSIS | <pre>STATUS vmContextDelete (VM_CONTEXT_ID context)</pre> |
| DESCRIPTION | <p>This routine deallocates the underlying translation table associated with a virtual memory context. It does not free the physical memory already mapped to the virtual memory space.</p> <p>This routine should not be called from interrupt level.</p> |
| AVAILABILITY | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI. |
| RETURNS | OK, or ERROR if <i>context</i> is not a valid context descriptor or if an error occurs deleting the translation table. |
| SEE ALSO | <code>vmLib</code> |

vmContextShow()

| | |
|---------------------|---|
| NAME | <i>vmContextShow()</i> – display the translation table for a context (VxVMI Opt.) |
| SYNOPSIS | <pre>STATUS vmContextShow (VM_CONTEXT_ID context /* context - NULL == currentContext */)</pre> |
| DESCRIPTION | <p>This routine displays the translation table for a specified context. If <i>context</i> is specified as NULL, the current context is displayed. Output is formatted to show blocks of virtual memory with consecutive physical addresses and the same state. State information shows the writable and cacheable states. If the block is in global virtual memory, the word "global" is appended to the line. Only virtual memory that has its valid state bit set is displayed.</p> <p>This routine should be used for debugging purposes only.</p> <p>Note that this routine cannot report non-standard architecture-dependent states.</p> |
| AVAILABILITY | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI. |
| RETURNS | OK, or ERROR if the virtual memory context is invalid. |
| SEE ALSO | vmShow |

vmCurrentGet()

| | |
|---------------------|---|
| NAME | <i>vmCurrentGet()</i> – get the current virtual memory context (VxVMI Opt.) |
| SYNOPSIS | <pre>VM_CONTEXT_ID vmCurrentGet (void)</pre> |
| DESCRIPTION | <p>This routine returns the current virtual memory context.</p> <p>This routine is callable from interrupt level.</p> |
| AVAILABILITY | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI. |
| RETURNS | The current virtual memory context, or NULL if no virtual memory context is installed. |
| SEE ALSO | vmLib |

vmCurrentSet()

| | |
|---------------------|---|
| NAME | <i>vmCurrentSet()</i> – set the current virtual memory context (VxVMI Opt.) |
| SYNOPSIS | <pre>STATUS vmCurrentSet (VM_CONTEXT_ID context /* context to install */)</pre> |
| DESCRIPTION | This routine installs a specified virtual memory context. This routine is callable from interrupt level. |
| AVAILABILITY | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI. |
| RETURNS | OK, or ERROR if the validation or context switch fails. |
| SEE ALSO | <i>vmLib</i> |

vmEnable()

| | |
|---------------------|--|
| NAME | <i>vmEnable()</i> – enable or disable virtual memory (VxVMI Opt.) |
| SYNOPSIS | <pre>STATUS vmEnable (BOOL enable /* TRUE == enable MMU, FALSE == disable MMU */)</pre> |
| DESCRIPTION | This routine turns virtual memory on and off. Memory management should not be turned off once it is turned on except in the case of system shutdown. This routine is callable from interrupt level. |
| AVAILABILITY | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI. |
| RETURNS | OK, or ERROR if the validation or architecture-dependent code fails. |
| SEE ALSO | <i>vmLib</i> |

vmGlobalInfoGet()

NAME *vmGlobalInfoGet()* – get global virtual memory information (VxVMI Opt.)

SYNOPSIS `UINT8 *vmGlobalInfoGet (void)`

DESCRIPTION This routine provides a description of those parts of the virtual memory space dedicated to global memory. The routine returns a pointer to an array of `UINT8`. Each element of the array corresponds to a block of virtual memory, the size of which is architecture-dependent and can be obtained with a call to *vmPageBlockSizeGet()*. To determine if a particular address is in global virtual memory, use the following code:

```
UINT8 *globalPageBlockArray = vmGlobalInfoGet ();
int pageBlockSize = vmPageBlockSizeGet ();

if (globalPageBlockArray[addr/pageBlockSize])
    ...
```

The array pointed to by the returned pointer is guaranteed to be static as long as no calls are made to *vmGlobalMap()* while the array is being examined. The information in the array can be used to determine what portions of the virtual memory space are available for use as private virtual memory within a virtual memory context.

This routine is callable from interrupt level.

AVAILABILITY This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.

RETURNS A pointer to an array of `UINT8`.

SEE ALSO `vmLib`, *vmPageBlockSizeGet()*

vmGlobalMap()

NAME *vmGlobalMap()* – map physical pages to virtual space in shared global virtual memory (VxVMI Opt.)

SYNOPSIS `STATUS vmGlobalMap`
`(`
`void * virtualAddr, /* virtual address */`
`void * physicalAddr, /* physical address */`
`UINT len /* len of virtual and physical spaces */`
`)`

- DESCRIPTION** This routine maps physical pages to virtual space that is shared by all virtual memory contexts. Calls to *vmGlobalMap()* should be made before any virtual memory contexts are created to insure that the shared global mappings are included in all virtual memory contexts. Mappings created with *vmGlobalMap()* after virtual memory contexts are created are not guaranteed to appear in all virtual memory contexts. After the call to *vmGlobalMap()*, the state of all pages in the the newly mapped virtual memory is unspecified and must be set with a call to *vmStateSet()*, once the initial virtual memory context is created.
- This routine should not be called from interrupt level.
- AVAILABILITY** This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
- RETURNS** OK, or ERROR if *virtualAddr* or *physicalAddr* are not on page boundaries, *len* is not a multiple of the page size, or the mapping fails.
- ERRNO** S_vmLib_NOT_PAGE_ALIGNED
- SEE ALSO** vmLib

vmGlobalMapInit()

- NAME** *vmGlobalMapInit()* – initialize global mapping (VxVMI Opt.)
- SYNOPSIS**
- ```
VM_CONTEXT_ID vmGlobalMapInit
(
 PHYS_MEM_DESC * pMemDescArray, /* pointer to array of mem desc */
 int numDescArrayElements, /* # of elements in pMemDescArray */
 BOOL enable /* enable virtual memory */
)
```
- DESCRIPTION** This routine is a convenience routine that creates and installs a virtual memory context with global mappings defined for each contiguous memory segment defined in the physical memory descriptor array passed as an argument. The context ID returned becomes the current virtual memory context.
- The physical memory descriptor also contains state information used to initialize the state information in the MMU's translation table for that memory segment. The following state bits may be or'ed together:
- |                    |                        |                          |
|--------------------|------------------------|--------------------------|
| VM_STATE_VALID     | VM_STATE_VALID_NOT     | valid/invalid            |
| VM_STATE_WRITABLE  | VM_STATE_WRITABLE_NOT  | writable/write-protected |
| VM_STATE_CACHEABLE | VM_STATE_CACHEABLE_NOT | cacheable/not-cacheable  |



Additionally, mask bits are or'ed together in the **initialStateMask** structure element to describe which state bits are being specified in the **initialState** structure element:

```
VM_STATE_MASK_VALID
VM_STATE_MASK_WRITABLE
VM_STATE_MASK_CACHEABLE
```

If the *enable* parameter is TRUE, the MMU is enabled upon return. The *vmGlobalMapInit()* routine should be called only after *vmLibInit()* has been called.

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI. |
| <b>RETURNS</b>      | A pointer to a newly created virtual memory context, or NULL if the memory cannot be mapped.      |
| <b>SEE ALSO</b>     | <b>vmLib</b>                                                                                      |

---

## *vmLibInit()*

|                     |                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <i>vmLibInit()</i> – initialize the virtual memory support module (VxVMI Opt.)                                  |
| <b>SYNOPSIS</b>     | <pre>STATUS vmLibInit<br/>(<br/>    int pageSize /* size of page */<br/>)</pre>                                 |
| <b>DESCRIPTION</b>  | This routine initializes the virtual memory context class. It is called only once during system initialization. |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.               |
| <b>RETURNS</b>      | OK.                                                                                                             |
| <b>SEE ALSO</b>     | <b>vmLib</b>                                                                                                    |

**vmMap()**

---

**vmMap()****NAME** `vmMap()` – map physical space into virtual space (VxVMI Opt.)

**SYNOPSIS**

```

STATUS vmMap
(
 VM_CONTEXT_ID context, /* context - NULL == currentContext */
 void * virtualAddr, /* virtual address */
 void * physicalAddr, /* physical address */
 UINT len /* len of virtual and physical spaces */
)

```

**DESCRIPTION**

This routine maps physical pages into a contiguous block of virtual memory. *virtualAddr* and *physicalAddr* must be on page boundaries, and *len* must be evenly divisible by the page size. After the call to `vmMap()`, the state of all pages in the newly mapped virtual memory is valid, writable, and cacheable.

The `vmMap()` routine can fail if the specified virtual address space conflicts with the translation tables of the global virtual memory space. The global virtual address space is architecture-dependent and is initialized at boot time with calls to `vmGlobalMap()` by `vmGlobalMapInit()`. If a conflict results, `errno` is set to `S_vmLib_ADDR_IN_GLOBAL_SPACE`. To avoid this conflict, use `vmGlobalInfoGet()` to ascertain which portions of the virtual address space are reserved for the global virtual address space. If *context* is specified as `NULL`, the current virtual memory context is used.

This routine should not be called from interrupt level.

**AVAILABILITY** This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.

**RETURNS** OK, or ERROR if *virtualAddr* or *physicalAddr* are not on page boundaries, *len* is not a multiple of the page size, the validation fails, or the mapping fails.

**ERRNO** `S_vmLib_NOT_PAGE_ALIGNED`, `S_vmLib_ADDR_IN_GLOBAL_SPACE`

**SEE ALSO** `vmLib`

---

## *vmPageBlockSizeGet()*

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <i>vmPageBlockSizeGet()</i> – get the architecture-dependent page block size (VxVMI Opt.)                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>     | <code>int vmPageBlockSizeGet (void)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b>  | <p>This routine returns the size of a page block for the current architecture. Each MMU architecture constructs translation tables such that a minimum number of pages are pre-defined when a new section of the translation table is built. This minimal group of pages is referred to as a "page block." This routine may be used in conjunction with <i>vmGlobalInfoGet()</i> to examine the layout of global virtual memory.</p> <p>This routine is callable from interrupt level.</p> |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>RETURNS</b>      | The page block size of the current architecture.                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>     | <i>vmLib</i> , <i>vmGlobalInfoGet()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

---

## *vmPageSizeGet()*

|                     |                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <i>vmPageSizeGet()</i> – return the page size (VxVMI Opt.)                                                              |
| <b>SYNOPSIS</b>     | <code>int vmPageSizeGet (void)</code>                                                                                   |
| <b>DESCRIPTION</b>  | <p>This routine returns the architecture-dependent page size.</p> <p>This routine is callable from interrupt level.</p> |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.                       |
| <b>RETURNS</b>      | The page size of the current architecture.                                                                              |
| <b>SEE ALSO</b>     | <i>vmLib</i>                                                                                                            |

---

## *vmShowInit()*

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <i>vmShowInit()</i> – include virtual memory show facility (VxVMI Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>     | <code>void vmShowInit (void)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b>  | <p>This routine acts as a hook to include <i>vmContextShow()</i>. It is called automatically when the virtual memory show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none"><li>– If you use the configuration header files, define both <code>INCLUDE_MMU_FULL</code> and <code>INCLUDE_SHOW_ROUTINES</code> in <code>config.h</code>.</li><li>– If you use the Tornado project facility, select <code>INCLUDE_MMU_FULL_SHOW</code>.</li></ul> |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>RETURNS</b>      | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>     | <code>vmShow</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

---

## *vmStateGet()*

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                             |                                 |               |                                |                                    |                          |                                 |                                     |                         |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|---------------------------------|---------------|--------------------------------|------------------------------------|--------------------------|---------------------------------|-------------------------------------|-------------------------|
| <b>NAME</b>                     | <i>vmStateGet()</i> – get the state of a page of virtual memory (VxVMI Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                             |                                 |               |                                |                                    |                          |                                 |                                     |                         |
| <b>SYNOPSIS</b>                 | <pre>STATUS vmStateGet (     VM_CONTEXT_ID context,    /* context - NULL == currentContext */     void *          pPageAddr, /* virtual page addr */     UINT *          pState     /* where to return state */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                 |                             |                                 |               |                                |                                    |                          |                                 |                                     |                         |
| <b>DESCRIPTION</b>              | <p>This routine extracts state bits with the following masks:</p> <p><code>VM_STATE_MASK_VALID</code><br/><code>VM_STATE_MASK_WRITABLE</code><br/><code>VM_STATE_MASK_CACHEABLE</code></p> <p>Individual states may be identified with the following constants:</p> <table><tr><td><code>VM_STATE_VALID</code></td><td><code>VM_STATE_VALID_NOT</code></td><td>valid/invalid</td></tr><tr><td><code>VM_STATE_WRITABLE</code></td><td><code>VM_STATE_WRITABLE_NOT</code></td><td>writable/write-protected</td></tr><tr><td><code>VM_STATE_CACHEABLE</code></td><td><code>VM_STATE_CACHEABLE_NOT</code></td><td>cacheable/not-cacheable</td></tr></table> | <code>VM_STATE_VALID</code> | <code>VM_STATE_VALID_NOT</code> | valid/invalid | <code>VM_STATE_WRITABLE</code> | <code>VM_STATE_WRITABLE_NOT</code> | writable/write-protected | <code>VM_STATE_CACHEABLE</code> | <code>VM_STATE_CACHEABLE_NOT</code> | cacheable/not-cacheable |
| <code>VM_STATE_VALID</code>     | <code>VM_STATE_VALID_NOT</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | valid/invalid               |                                 |               |                                |                                    |                          |                                 |                                     |                         |
| <code>VM_STATE_WRITABLE</code>  | <code>VM_STATE_WRITABLE_NOT</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | writable/write-protected    |                                 |               |                                |                                    |                          |                                 |                                     |                         |
| <code>VM_STATE_CACHEABLE</code> | <code>VM_STATE_CACHEABLE_NOT</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | cacheable/not-cacheable     |                                 |               |                                |                                    |                          |                                 |                                     |                         |

For example, to see if a page is writable, the following code would be used:

```
vmStateGet (vmContext, pageAddr, &state);
if ((state & VM_STATE_MASK_WRITABLE) & VM_STATE_WRITABLE)
 ...
```

If *context* is specified as NULL, the current virtual memory context is used.

This routine is callable from interrupt level.

|                     |                                                                                                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.                                                                     |
| <b>RETURNS</b>      | OK, or ERROR if <i>pageAddr</i> is not on a page boundary, the validity check fails, or the architecture-dependent state get fails for the specified virtual address. |
| <b>ERRNO</b>        | S_vmLib_NOT_PAGE_ALIGNED                                                                                                                                              |
| <b>SEE ALSO</b>     | vmLib                                                                                                                                                                 |

---

## *vmStateSet()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>vmStateSet()</i> – change the state of a block of virtual memory (VxVMI Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS vmStateSet<br/>(<br/>    VM_CONTEXT_ID context, /* context - NULL == currentContext */<br/>    void * pVirtual, /* virtual address to modify state of */<br/>    int len, /* len of virtual space to modify state of */<br/>    UINT stateMask, /* state mask */<br/>    UINT state /* state */<br/>)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>DESCRIPTION</b> | This routine changes the state of a block of virtual memory. Each page of virtual memory has at least three elements of state information: validity, writability, and cacheability. Specific architectures may define additional state information; see <b>vmLib.h</b> for additional architecture-specific states. Memory accesses to a page marked as invalid will result in an exception. Pages may be invalidated to prevent them from being corrupted by invalid references. Pages may be defined as read-only or writable, depending on the state of the writable bits. Memory accesses to pages marked as not-cacheable will always result in a memory cycle, bypassing the cache. This is useful for multiprocessing, multiple bus masters, and hardware control registers. |

The following states are provided and may be or'ed together in the state parameter:

|                    |                        |                          |
|--------------------|------------------------|--------------------------|
| VM_STATE_VALID     | VM_STATE_VALID_NOT     | valid/invalid            |
| VM_STATE_WRITABLE  | VM_STATE_WRITABLE_NOT  | writable/write-protected |
| VM_STATE_CACHEABLE | VM_STATE_CACHEABLE_NOT | cacheable/not-cacheable  |

Additionally, the following masks are provided so that only specific states may be set. These may be or'ed together in the **stateMask** parameter.

VM\_STATE\_MASK\_VALID  
VM\_STATE\_MASK\_WRITABLE  
VM\_STATE\_MASK\_CACHEABLE

If *context* is specified as NULL, the current context is used.

This routine is callable from interrupt level.

|              |                                                                                                                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AVAILABILITY | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.                                                                                                            |
| RETURNS      | OK or, ERROR if the validation fails, <i>pVirtual</i> is not on a page boundary, <i>len</i> is not a multiple of page size, or the architecture-dependent state set fails for the specified virtual address. |
| ERRNO        | S_vmLib_NOT_PAGE_ALIGNED, S_vmLib_BAD_STATE_PARAM,<br>S_vmLib_BAD_MASK_PARAM                                                                                                                                 |
| SEE ALSO     | vmLib                                                                                                                                                                                                        |

---

## ***vmTextProtect()***

|              |                                                                                                                                                                                                                                                                    |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NAME         | <i>vmTextProtect()</i> – write-protect a text segment (VxVMI Opt.)                                                                                                                                                                                                 |
| SYNOPSIS     | <b>STATUS</b> <i>vmTextProtect</i> (void)                                                                                                                                                                                                                          |
| DESCRIPTION  | This routine write-protects the VxWorks text segment and sets a flag so that all text segments loaded by the incremental loader will be write-protected. The routine should be called after both <i>vmLibInit()</i> and <i>vmGlobalMapInit()</i> have been called. |
| AVAILABILITY | This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.                                                                                                                                                                  |
| RETURNS      | OK, or ERROR if the text segment cannot be write-protected.                                                                                                                                                                                                        |

**ERRNO** S\_vmLib\_TEXT\_PROTECTION\_UNAVAILABLE

**SEE ALSO** vmLib

2

---

## *vmTranslate()*

**NAME** *vmTranslate()* – translate a virtual address to a physical address (VxVMI Opt.)

**SYNOPSIS**

```
STATUS vmTranslate
(
 VM_CONTEXT_ID context, /* context - NULL == currentContext */
 void * virtualAddr, /* virtual address */
 void * *physicalAddr /* place to put result */
)
```

**DESCRIPTION** This routine retrieves mapping information for a virtual address from the page translation tables. If the specified virtual address has never been mapped, the returned status can be either OK or ERROR; however, if it is OK, then the returned physical address will be -1. If *context* is specified as NULL, the current context is used.

This routine is callable from interrupt level.

**AVAILABILITY** This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.

**RETURNS** OK, or ERROR if the validation or translation fails.

**SEE ALSO** vmLib

---

## *vprintf()*

**NAME** *vprintf()* – write a string formatted with a variable argument list to standard output (ANSI)

**SYNOPSIS**

```
int vprintf
(
 const char * fmt, /* format string to write */
 va_list vaList /* arguments to format */
)
```

***vsprintf()***

- DESCRIPTION** This routine prints a string formatted with a variable argument list to standard output. It is identical to *printf()*, except that it takes the variable arguments to be formatted as a list *vaList* of type **va\_list** rather than as in-line arguments.
- RETURNS** The number of characters output, or ERROR if there is an error during output.
- SEE ALSO** *fcntlLib, printf()*, *American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: Input/Output (stdio.h)*

***vsprintf()***

- NAME** *vsprintf()* – write a string formatted with a variable argument list to a buffer (ANSI)
- SYNOPSIS**
- ```
int vsprintf
(
    char *      buffer, /* buffer to write to */
    const char * fmt,  /* format string */
    va_list     vaList /* optional arguments to format */
)
```
- DESCRIPTION** This routine copies a string formatted with a variable argument list to a specified buffer. This routine is identical to *sprintf()*, except that it takes the variable arguments to be formatted as a list *vaList* of type **va_list** rather than as in-line arguments.
- RETURNS** The number of characters copied to *buffer*, not including the NULL terminator.
- SEE ALSO** *fcntlLib, sprintf()*, *American National Standard for Information Systems – Programming Language – C, ANSI X3.159-1989: Input/Output (stdio.h)*

vxMemArchProbe()

- NAME** *vxMemArchProbe()* – architecture specific part of *vxMemProbe*
- SYNOPSIS**
- ```
STATUS vxMemArchProbe
(
 char * adrs, /* address to be probed */
 int mode, /* VX_READ or VX_WRITE */
 int length, /* 1, 2, 4, or 8 */
 char * pVal /* where to return value, or ptr to value to be written */
)
```



|                    |                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This is the routine implementing the architecture specific part of the vxMemProbe routine. It traps the relevant exceptions while accessing the specified address. If an exception occurs, then the result will be ERROR. If no exception occurs then the result will be OK. |
| <b>RETURNS</b>     | OK or ERROR if an exception occurred during access.                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | vxLib                                                                                                                                                                                                                                                                        |

---

## *vxMemProbe()*

**NAME** *vxMemProbe()* – probe an address for a bus error

**SYNOPSIS**

```
STATUS vxMemProbe
(
 char * adrs, /* address to be probed */
 int mode, /* VX_READ or VX_WRITE */
 int length, /* 1, 2, 4, or 8 */
 char * pVal /* where to return value, or ptr to value to be written */
)
```

**DESCRIPTION** This routine probes a specified address to see if it is readable or writable, as specified by *mode*. The address is read or written as 1, 2, or 4 bytes, as specified by *length* (values other than 1, 2, or 4 yield unpredictable results). If the probe is a VX\_READ (0), the value read is copied to the location pointed to by *pVal*. If the probe is a VX\_WRITE (1), the value written is taken from the location pointed to by *pVal*. In either case, *pVal* should point to a value of 1, 2, or 4 bytes, as specified by *length*.

Note that only bus errors are trapped during the probe, and that the access must otherwise be valid (i.e., it must not generate an address error).

**EXAMPLE**

```
testMem (adrs)
 char *adrs;
 {
 char testW = 1;
 char testR;
 if (vxMemProbe (adrs, VX_WRITE, 1, &testW) == OK)
 printf ("value %d written to adrs %x\n", testW, adrs);
 if (vxMemProbe (adrs, VX_READ, 1, &testR) == OK)
 printf ("value %d read from adrs %x\n", testR, adrs);
 }
```

***vxMemProbeAsi()***

- MODIFICATION** The BSP can modify the behaviour of *vxMemProbe()* by supplying an alternate routine and placing the address in the global variable `_func_vxMemProbeHook`. The BSP routine will be called instead of the architecture specific routine *vxMemArchProbe()*.
- RETURNS** OK, or ERROR if the probe caused a bus error or was misaligned.
- SEE ALSO** *vxLib*, *vxMemArchProbe()*

---

## *vxMemProbeAsi()*

**NAME** *vxMemProbeAsi()* – probe address in ASI space for bus error (SPARC)

**SYNOPSIS**

```
STATUS vxMemProbeAsi
(
 char * adrs, /* address to be probed */
 int mode, /* VX_READ or VX_WRITE */
 int length, /* 1, 2, 4, or 8 */
 char * pVal, /* where to return value, or ptr to value to be written */
 int adrsAsi /* ASI field of address to be probed */
)
```

**DESCRIPTION** This routine probes the specified address to see if it is readable or writable, as specified by *mode*. The address will be read/written as 1, 2, 4, or 8 bytes as specified by *length* (values other than 1, 2, 4, or 8 return ERROR). If the probe is a `VX_READ` (0), then the value read will be returned in the location pointed to by *pVal*. If the probe is a `VX_WRITE` (1), then the value written will be taken from the location pointed to by *pVal*. In either case, *pVal* should point to a value of the appropriate length, 1, 2, 4, or 8 bytes, as specified by *length*.

The fifth parameter *adrsAsi* is the ASI parameter used to modify the *adrs* parameter.

**EXAMPLE**

```
testMem (adrs)
 char *adrs;
 {
 char testW = 1;
 char testR;
 if (vxMemProbeAsi (adrs, VX_WRITE, 1, &testW) == OK)
 printf ("value %d written to adrs %x\n", testW, adrs);
 if (vxMemProbeAsi (adrs, VX_READ, 1, &testR) == OK)
 printf ("value %d read from adrs %x\n", testR, adrs);
 }
```

**RETURNS** OK, or ERROR if the probe caused a bus error or was misaligned.

**SEE ALSO** *vxLib*

---

## ***vxPowerDown()***

|                    |                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>vxPowerDown()</i> – place the processor in reduced-power mode (PowerPC)                                                                                                                                           |
| <b>SYNOPSIS</b>    | UINT32 <i>vxPowerDown</i> (void)                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | This routine activates the reduced-power mode if power management is enabled. It is called by the scheduler when the kernel enters the idle loop. The power management mode is selected by <i>vxPowerModeSet()</i> . |
| <b>RETURNS</b>     | OK, or ERROR if power management is not supported or if external interrupts are disabled.                                                                                                                            |
| <b>SEE ALSO</b>    | <i>vxLib</i> , <i>vxPowerModeSet()</i> , <i>vxPowerModeGet()</i><br>STATUS <i>vxPowerDown</i> (void)                                                                                                                 |

---

## ***vxPowerModeGet()***

|                    |                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>vxPowerModeGet()</i> – get the power management mode (PowerPC)                                         |
| <b>SYNOPSIS</b>    | UINT32 <i>vxPowerModeGet</i> (void)                                                                       |
| <b>DESCRIPTION</b> | This routine returns the power management mode set by <i>vxPowerModeSet()</i> .                           |
| <b>RETURNS</b>     | The power management mode, or ERROR if no mode has been selected or if power management is not supported. |
| <b>SEE ALSO</b>    | <i>vxLib</i> , <i>vxPowerModeSet()</i> , <i>vxPowerDown()</i>                                             |

---

## ***vxPowerModeSet()***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>vxPowerModeSet()</i> – set the power management mode (PowerPC)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS vxPowerModeSet (     UINT32 mode /* power management mode to select */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | <p>This routine selects the power management mode to be activated when <i>vxPowerDown()</i> is called. <i>vxPowerModeSet()</i> is normally called in the BSP initialization routine <i>sysHwInit()</i>.</p> <p>Power management modes include the following:</p> <p><b>VX_POWER_MODE_DISABLE</b> (0x1)<br/>Power management is disabled; this prevents the MSR(POW) bit from being set (all PPC).</p> <p><b>VX_POWER_MODE_FULL</b> (0x2)<br/>All CPU units are active while the kernel is idle (PPC603, PPCEC603 and PPC860 only).</p> <p><b>VX_POWER_MODE_DOZE</b> (0x4)<br/>Only the decremter, data cache, and bus snooping are active while the kernel is idle (PPC603, PPCEC603 and PPC860).</p> <p><b>VX_POWER_MODE_NAP</b> (0x8)<br/>Only the decremter is active while the kernel is idle (PPC603, PPCEC603 and PPC604).</p> <p><b>VX_POWER_MODE_SLEEP</b> (0x10)<br/>All CPU units are inactive while the kernel is idle (PPC603, PPCEC603 and PPC860 – not recommended for the PPC603 and PPCEC603 architecture).</p> <p><b>VX_POWER_MODE_DEEP_SLEEP</b> (0x20)<br/>All CPU units are inactive while the kernel is idle (PPC860 only – not recommended).</p> <p><b>VX_POWER_MODE_DPM</b> (0x40)<br/>Dynamic Power Management Mode (PPC603 and PPCEC603 only).</p> <p><b>VX_POWER_MODE_DOWN</b> (0x80)<br/>Only a hard reset causes an exit from power-down low power mode (PPC860 only – not recommended).</p> |
| <b>RETURNS</b>     | OK, or ERROR if <i>mode</i> is incorrect or not supported by the processor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <i>vxLib</i> , <i>vxPowerModeGet()</i> , <i>vxPowerDown()</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

---

## *vxSSDisable()*

|                    |                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>vxSSDisable()</i> – disable the superscalar dispatch (MC68060)                                                   |
| <b>SYNOPSIS</b>    | <code>void vxSSDisable (void)</code>                                                                                |
| <b>DESCRIPTION</b> | This function resets the ESS bit of the Processor Configuration Register (PCR) to disable the superscalar dispatch. |
| <b>RETURNS</b>     | N/A                                                                                                                 |
| <b>SEE ALSO</b>    | <code>vxLib</code>                                                                                                  |

---

## *vxSSEnable()*

|                    |                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>vxSSEnable()</i> – enable the superscalar dispatch (MC68060)                                                  |
| <b>SYNOPSIS</b>    | <code>void vxSSEnable (void)</code>                                                                              |
| <b>DESCRIPTION</b> | This function sets the ESS bit of the Processor Configuration Register (PCR) to enable the superscalar dispatch. |
| <b>RETURNS</b>     | N/A                                                                                                              |
| <b>SEE ALSO</b>    | <code>vxLib</code>                                                                                               |

---

## *vxTas()*

|                    |                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>vxTas()</i> – C-callable atomic test-and-set primitive                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>BOOL vxTas (     void * address /* address to test and set */ )</pre>                                                                                                      |
| <b>DESCRIPTION</b> | This routine provides a C-callable interface to a test-and-set instruction. The instruction is executed on the specified address. The architecture test-and-set instruction is: |

**VXWBSem::VXWBSem( )**

68K: **tas**  
 SPARC: **ldstub**  
 i960: **atmod**  
 ARM **swpb**

This routine is equivalent to *sysBusTas( )* in *sysLib*.

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <b>BUGS (MIPS)</b> | Only <b>Kseg0</b> and <b>Kseg1</b> addresses are accepted; other addresses always return FALSE. |
| <b>RETURNS</b>     | TRUE if the value had not been set (but is now), or FALSE if the value was set already.         |
| <b>SEE ALSO</b>    | <i>vxLib</i> , <i>sysBusTas( )</i>                                                              |

---

## VXWBSem::VXWBSem( )

**NAME** *VXWBSem::VXWBSem( )* – create and initialize a binary semaphore (WFC Opt.)

**SYNOPSIS**

```
VXWBSem
(
 int opts,
 SEM_B_STATE iState
)
```

**DESCRIPTION** This routine allocates and initializes a binary semaphore. The semaphore is initialized to the state *iState*: either **SEM\_FULL** (1) or **SEM\_EMPTY** (0).

The *opts* parameter specifies the queuing style for blocked tasks. Tasks can be queued on a priority basis or a first-in-first-out basis. These options are **SEM\_Q\_PRIORITY** and **SEM\_Q\_FIFO**, respectively.

Binary semaphores are the most versatile, efficient, and conceptually simple type of semaphore. They can be used to: (1) control mutually exclusive access to shared devices or data structures, or (2) synchronize multiple tasks, or task-level and interrupt-level processes. Binary semaphores form the foundation of numerous VxWorks facilities.

A binary semaphore can be viewed as a cell in memory whose contents are in one of two states, full or empty. When a task takes a binary semaphore, using *VXWSem::take( )*, subsequent action depends on the state of the semaphore:

- (1) If the semaphore is full, the semaphore is made empty, and the calling task continues executing.
- (2) If the semaphore is empty, the task is blocked, pending the availability of the semaphore. If a timeout is specified and the timeout expires, the pended task is removed from the queue of pended tasks and enters the ready state with an **ERROR**

status. A pended task is ineligible for CPU allocation. Any number of tasks may be pended simultaneously on the same binary semaphore.

When a task gives a binary semaphore, using **VXWSem::give()**, the next available task in the pend queue is unblocked. If no task is pending on this semaphore, the semaphore becomes full. Note that if a semaphore is given, and a task is unblocked that is of higher priority than the task that called **VXWSem::give()**, the unblocked task preempts the calling task.

#### MUTUAL EXCLUSION

To use a binary semaphore as a means of mutual exclusion, first create it with an initial state of full.

Then guard a critical section or resource by taking the semaphore with **VXWSem::take()**, and exit the section or release the resource by giving the semaphore with **VXWSem::give()**.

While there is no restriction on the same semaphore being given, taken, or flushed by multiple tasks, it is important to ensure the proper functionality of the mutual-exclusion construct. While there is no danger in any number of processes taking a semaphore, the giving of a semaphore should be more carefully controlled. If a semaphore is given by a task that did not take it, mutual exclusion could be lost.

#### SYNCHRONIZATION

To use a binary semaphore as a means of synchronization, create it with an initial state of empty. A task blocks by taking a semaphore at a synchronization point, and it remains blocked until the semaphore is given by another task or interrupt service routine.

Synchronization with interrupt service routines is a particularly common need. Binary semaphores can be given, but not taken, from interrupt level. Thus, a task can block at a synchronization point with **VXWSem::take()**, and an interrupt service routine can unblock that task with **VXWSem::give()**.

A **semFlush()** on a binary semaphore atomically unblocks all pended tasks in the semaphore queue; that is, all tasks are unblocked at once, before any actually execute.

#### CAVEATS

There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible. Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus, if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores will not be given back, effectively leaving a resource permanently unavailable. The mutual-exclusion semaphores provided by **VXWMSem** offer protection from unexpected task deletion.

#### RETURNS

N/A

#### SEE ALSO

**VXWSem**

---

## VXWCSem::VXWCSem( )

**NAME** VXWCSem::VXWCSem() – create and initialize a counting semaphore (WFC Opt.)

**SYNOPSIS**

```
VXWCSem
(
 int opts,
 int count
)
```

**DESCRIPTION** This routine allocates and initializes a counting semaphore. The semaphore is initialized to the specified initial count.

The *opts* parameter specifies the queuing style for blocked tasks. Tasks may be queued on a priority basis or a first-in-first-out basis. These options are **SEM\_Q\_PRIORITY** and **SEM\_Q\_FIFO**, respectively.

A counting semaphore may be viewed as a cell in memory whose contents keep track of a count. When a task takes a counting semaphore, using *VXWSem::take()*, subsequent action depends on the state of the count:

- (1) If the count is non-zero, it is decremented and the calling task continues executing.
- (2) If the count is zero, the task is blocked, pending the availability of the semaphore. If a timeout is specified and the timeout expires, the pended task is removed from the queue of pended tasks and enters the ready state with an **ERROR** status. A pended task is ineligible for CPU allocation. Any number of tasks may be pended simultaneously on the same counting semaphore.

When a task gives a semaphore, using *VXWSem::give()*, the next available task in the pend queue is unblocked. If no task is pending on this semaphore, the semaphore count is incremented. Note that if a semaphore is given, and a task is unblocked that is of higher priority than the task that called *VXWSem::give()*, the unblocked task preempts the calling task.

A *VXWSem::flush()* on a counting semaphore atomically unblocks all pended tasks in the semaphore queue. Thus, all tasks are made ready before any task actually executes. The count of the semaphore remains unchanged.

**INTERRUPT USAGE** Counting semaphores may be given but not taken from interrupt level.

**CAVEATS** There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible. Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus, if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores are not given back, effectively leaving a resource permanently unavailable. The



mutual-exclusion semaphores provided by **VXWMSem** offer protection from unexpected task deletion.

**RETURNS** N/A

**SEE ALSO** **VXWSem**

---

## ***VXWList::add()***

**NAME** ***VXWList::add()*** – add a node to the end of list (WFC Opt.)

**SYNOPSIS**

```
void add
(
 NODE * pNode
)
```

**DESCRIPTION** This routine adds a specified node to the end of the list.

**RETURNS** N/A

**SEE ALSO** **VXWList**

---

## ***VXWList::concat()***

**NAME** ***VXWList::concat()*** – concatenate two lists (WFC Opt.)

**SYNOPSIS**

```
void concat
(
 VXWList &aList
)
```

**DESCRIPTION** This routine concatenates the specified list to the end of the current list. The specified list is left empty. Either list (or both) can be empty at the beginning of the operation.

**RETURNS** N/A

**SEE ALSO** **VXWList**

---

## ***VXWList::count()***

**NAME** *VXWList::count()* – report the number of nodes in a list (WFC Opt.)

**SYNOPSIS** `int count ()`

**DESCRIPTION** This routine returns the number of nodes in a specified list.

**RETURNS** The number of nodes in the list.

**SEE ALSO** *VXWList*

---

## ***VXWList::extract()***

**NAME** *VXWList::extract()* – extract a sublist from list (WFC Opt.)

**SYNOPSIS** `LIST extract  
(  
    NODE * pStart,  
    NODE * pEnd  
)`

**DESCRIPTION** This routine extracts the sublist that starts with *pStart* and ends with *pEnd*. It returns the extracted list.

**RETURNS** The extracted sublist.

**SEE ALSO** *VXWList*

---

## ***VXWList::find()***

**NAME** *VXWList::find()* – find a node in list (WFC Opt.)

**SYNOPSIS** `int find  
(  
    NODE * pNode  
) const`

**DESCRIPTION** This routine returns the node number of a specified node (the first node is 1).

**RETURNS** The node number, or ERROR if the node is not found.

**SEE ALSO** VXWList

---

## ***VXWList::first()***

**NAME** *VXWList::first()* – find first node in list (WFC Opt.)

**SYNOPSIS** `NODE * first ( )`

**DESCRIPTION** This routine finds the first node in its list.

**RETURNS** A pointer to the first node in the list, or NULL if the list is empty.

**SEE ALSO** VXWList

---

## ***VXWList::get()***

**NAME** *VXWList::get()* – delete and return the first node from list (WFC Opt.)

**SYNOPSIS** `NODE * get ( )`

**DESCRIPTION** This routine gets the first node from its list, deletes the node from the list, and returns a pointer to the node gotten.

**RETURNS** A pointer to the node gotten, or NULL if the list is empty.

**SEE ALSO** VXWList

***VXWList::insert()***

---

***VXWList::insert()***

**NAME** *VXWList::insert()* – insert a node in list after a specified node (WFC Opt.)

**SYNOPSIS**

```
void insert
(
 NODE * pPrev,
 NODE * pNode
)
```

**DESCRIPTION** This routine inserts a specified node into the list. The new node is placed following the list node *pPrev*. If *pPrev* is NULL, the node is inserted at the head of the list.

**RETURNS** N/A

**SEE ALSO** *VXWList*

---

***VXWList::last()***

**NAME** *VXWList::last()* – find the last node in list (WFC Opt.)

**SYNOPSIS**

```
NODE * last ()
```

**DESCRIPTION** This routine finds the last node in its list.

**RETURNS** A pointer to the last node in the list, or NULL if the list is empty.

**SEE ALSO** *VXWList*

---

***VXWList::next()***

**NAME** *VXWList::next()* – find the next node in list (WFC Opt.)

**SYNOPSIS**

```
NODE * next
(
 NODE * pNode
) const
```

**DESCRIPTION** This routine locates the node immediately following a specified node.

**RETURNS** A pointer to the next node in the list, or NULL if there is no next node.

**SEE ALSO** VXWList

---

## **VXWList::nStep()**

**NAME** **VXWList::nStep()** – find a list node *nStep* steps away from a specified node (WFC Opt.)

**SYNOPSIS**

```
NODE * nStep
(
 NODE * pNode,
 int nStep
) const
```

**DESCRIPTION** This routine locates the node *nStep* steps away in either direction from a specified node. If *nStep* is positive, it steps toward the tail. If *nStep* is negative, it steps toward the head. If the number of steps is out of range, NULL is returned.

**RETURNS** A pointer to the node *nStep* steps away, or NULL if the node is out of range.

**SEE ALSO** VXWList

---

## **VXWList::nth()**

**NAME** **VXWList::nth()** – find the Nth node in a list (WFC Opt.)

**SYNOPSIS**

```
NODE * nth
(
 int nodeNum
) const
```

**DESCRIPTION** This routine returns a pointer to the node specified *nodeNum* where the first node in the list is numbered 1. The search is optimized by searching forward from the beginning if the node is closer to the head, and searching back from the end if it is closer to the tail.

**RETURNS** A pointer to the Nth node, or NULL if there is no Nth node.

**SEE ALSO** VXWList

***VXWList::previous()***

---

## ***VXWList::previous()***

**NAME** *VXWList::previous()* – find the previous node in list (WFC Opt.)

**SYNOPSIS**

```
NODE * previous
(
 NODE * pNode
) const
```

**DESCRIPTION** This routine locates the node immediately preceding the node pointed to by *pNode*.

**RETURNS** A pointer to the previous node in the list, or NULL if there is no previous node.

**SEE ALSO** *VXWList*

---

## ***VXWList::remove()***

**NAME** *VXWList::remove()* – delete a specified node from list (WFC Opt.)

**SYNOPSIS**

```
void remove
(
 NODE * pNode
)
```

**DESCRIPTION** This routine deletes a specified node from its list.

**RETURNS** N/A

**SEE ALSO** *VXWList*

---

## ***VXWList::VXWList()***

**NAME** *VXWList::VXWList()* – initialize a list (WFC Opt.)

**SYNOPSIS**

```
VXWList ()
```

**DESCRIPTION** This constructor initializes a list as an empty list.

**RETURNS** N/A  
**SEE ALSO** `VXWList`

---

## *VXWList::VXWList()*

**NAME** `VXWList::VXWList()` – initialize a list as a copy of another (WFC Opt.)

**SYNOPSIS**

```
VXWList
(
 const VXWList &
)
```

**DESCRIPTION** This constructor builds a new list as a copy of an existing list.

**RETURNS** N/A

**SEE ALSO** `VXWList`

---

## *VXWList::~~VXWList()*

**NAME** `VXWList::~~VXWList()` – free up a list (WFC Opt.)

**SYNOPSIS**

```
~VXWList ()
```

**DESCRIPTION** This destructor frees up memory used for nodes.

**RETURNS** N/A

**SEE ALSO** `VXWList`

---

## ***VXWMemPart::addToPool()***

|                    |                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWMemPart::addToPool()</b> – add memory to a memory partition (WFC Opt.)                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS addToPool (     char * pool,     unsigned poolSize )</pre>                                                                          |
| <b>DESCRIPTION</b> | This routine adds memory to its memory partition. The new memory added need not be contiguous with memory previously assigned to the partition. |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                    |
| <b>SEE ALSO</b>    | VXWMemPart                                                                                                                                      |

---

## ***VXWMemPart::alignedAlloc()***

|                    |                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWMemPart::alignedAlloc()</b> – allocate aligned memory from partition (WFC Opt.)                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>void * alignedAlloc (     unsigned nBytes,     unsigned alignment )</pre>                                                                                                                                                                      |
| <b>DESCRIPTION</b> | This routine allocates a buffer of size <i>nBytes</i> from its partition. Additionally, it ensures that the allocated buffer begins on a memory address evenly divisible by <i>alignment</i> . The <i>alignment</i> parameter must be a power of 2. |
| <b>RETURNS</b>     | A pointer to the newly allocated block, or NULL if the buffer cannot be allocated.                                                                                                                                                                  |
| <b>SEE ALSO</b>    | VXWMemPart                                                                                                                                                                                                                                          |



---

## ***VXWMemPart::alloc()***

**NAME** *VXWMemPart::alloc()* – allocate a block of memory from partition (WFC Opt.)

**SYNOPSIS**

```
void * alloc
(
 unsigned nBytes
)
```

**DESCRIPTION** This routine allocates a block of memory from its partition. The size of the block allocated is equal to or greater than *nBytes*.

**RETURNS** A pointer to a block, or NULL if the call fails.

**SEE ALSO** *VXWMemPart::free()*

---

## ***VXWMemPart::findMax()***

**NAME** *VXWMemPart::findMax()* – find the size of the largest available free block (WFC Opt.)

**SYNOPSIS**

```
int findMax ()
```

**DESCRIPTION** This routine searches for the largest block in the memory partition free list and returns its size.

**RETURNS** The size, in bytes, of the largest available block.

**SEE ALSO** *VXWMemPart*

---

## ***VXWMemPart::free()***

**NAME** *VXWMemPart::free()* – free a block of memory in partition (WFC Opt.)

**SYNOPSIS**

```
STATUS free
(
 char * pBlock
)
```

***VXWMemPart::info()***

- DESCRIPTION** This routine returns to the partition's free memory list a block of memory previously allocated with *VXWMemPart::alloc()*.
- RETURNS** OK, or ERROR if the block is invalid.
- SEE ALSO** *VXWMemPart::alloc()*

---

***VXWMemPart::info()***

**NAME** *VXWMemPart::info()* – get partition information (WFC Opt.)

**SYNOPSIS**

```
STATUS info
(
 MEM_PART_STATS * pPartStats
) const
```

**DESCRIPTION** This routine takes a pointer to a MEM\_PART\_STATS structure. All the parameters of the structure are filled in with the current partition information.

**RETURNS** OK if the structure has valid data, otherwise ERROR.

**SEE ALSO** *VXWMemPart::show()*

---

***VXWMemPart::options()***

**NAME** *VXWMemPart::options()* – set the debug options for memory partition (WFC Opt.)

**SYNOPSIS**

```
STATUS options
(
 unsigned options
)
```

**DESCRIPTION** This routine sets the debug options for its memory partition. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, the error status is returned. There are four error-handling options that can be individually selected:

**MEM\_ALLOC\_ERROR\_LOG\_FLAG**

Log a message when there is an error in allocating memory.

**MEM\_ALLOC\_ERROR\_SUSPEND\_FLAG**

Suspend the task when there is an error in allocating memory (unless the task was spawned with the **VX\_UNBREAKABLE** option, in which case it cannot be suspended).

**MEM\_BLOCK\_ERROR\_LOG\_FLAG**

Log a message when there is an error in freeing memory.

**MEM\_BLOCK\_ERROR\_SUSPEND\_FLAG**

Suspend the task when there is an error in freeing memory (unless the task was spawned with the **VX\_UNBREAKABLE** option, in which case it cannot be suspended).

These options are discussed in detail in the library manual entry for **memLib**.

**RETURNS** OK or ERROR.

**SEE ALSO** **VXWMemPart**

---

## ***VXWMemPart::realloc()***

**NAME** ***VXWMemPart::realloc()*** – reallocate a block of memory in partition (WFC Opt.)

**SYNOPSIS**

```
void * realloc
(
 char * pBlock,
 int nBytes
)
```

**DESCRIPTION** This routine changes the size of a specified block of memory and returns a pointer to the new block. The contents that fit inside the new size (or old size if smaller) remain unchanged. The memory alignment of the new block is not guaranteed to be the same as the original block.

If *pBlock* is NULL, this call is equivalent to ***VXWMemPart::alloc()***.

**RETURNS** A pointer to the new block of memory, or NULL if the call fails.

**SEE ALSO** **VXWMemPart**

---

## VXWMemPart::show()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWMemPart::show()</b> – show partition blocks and statistics (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>STATUS show (     int type = 0 ) const</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine displays statistics about the available and allocated memory in its memory partition. It shows the number of bytes, the number of blocks, and the average block size in both free and allocated memory, and also the maximum block size of free memory. It also shows the number of blocks currently allocated and the average allocated block size.</p> <p>In addition, if <i>type</i> is 1, the routine displays a list of all the blocks in the free list of the specified partition.</p> |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>VXWMemPart</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

---

## VXWMemPart::VXWMemPart()

|                    |                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWMemPart::VXWMemPart()</b> – create a memory partition (WFC Opt.)                                                                                              |
| <b>SYNOPSIS</b>    | <pre>VXWMemPart (     char * pool,     unsigned poolSize )</pre>                                                                                                    |
| <b>DESCRIPTION</b> | <p>This constructor creates a new memory partition containing a specified memory pool. Partitions can be created to manage any number of separate memory pools.</p> |
| <b>NOTE</b>        | <p>The descriptor for the new partition is allocated out of the system memory partition (i.e., with <i>malloc()</i>).</p>                                           |
| <b>RETURNS</b>     | N/A.                                                                                                                                                                |
| <b>SEE ALSO</b>    | <b>VXWMemPart</b>                                                                                                                                                   |

---

## ***VXWModule::flags()***

**NAME** *VXWModule::flags()* – get the flags associated with this module (WFC Opt.)

**SYNOPSIS** `int flags ()`

**DESCRIPTION** This routine returns the flags associated with its module.

**RETURNS** The option flags.

**SEE ALSO** *VXWModule*

---

## ***VXWModule::info()***

**NAME** *VXWModule::info()* – get information about object module (WFC Opt.)

**SYNOPSIS** `STATUS info  
(  
MODULE_INFO * pModuleInfo  
) const`

**DESCRIPTION** This routine fills in a `MODULE_INFO` structure with information about the object module.

**RETURNS** OK or ERROR.

**SEE ALSO** *VXWModule*

---

## ***VXWModule::name()***

**NAME** *VXWModule::name()* – get the name associated with module (WFC Opt.)

**SYNOPSIS** `char * name ()`

**DESCRIPTION** This routine returns a pointer to the name associated with its module.

**RETURNS** A pointer to the module name.

**SEE ALSO** *VXWModule*

***VXWModule::segFirst()***

---

## ***VXWModule::segFirst()***

**NAME** *VXWModule::segFirst()* – find the first segment in module (WFC Opt.)

**SYNOPSIS** `SEGMENT_ID segFirst ()`

**DESCRIPTION** This routine returns information about the first segment of a module descriptor.

**RETURNS** A pointer to the segment ID.

**SEE ALSO** *VXWModule::segGet()*

---

## ***VXWModule::segGet()***

**NAME** *VXWModule::segGet()* – get (delete and return) the first segment from module (WFC Opt.)

**SYNOPSIS** `SEGMENT_ID segGet ()`

**DESCRIPTION** This routine returns information about the first segment of a module descriptor, and then deletes the segment from the module.

**RETURNS** A pointer to the segment ID, or NULL if the segment list is empty.

**SEE ALSO** *VXWModule::segFirst()*

---

## ***VXWModule::segNext()***

**NAME** *VXWModule::segNext()* – find the next segment in module (WFC Opt.)

**SYNOPSIS** `SEGMENT_ID segNext  
(  
    SEGMENT_ID segmentId  
) const`

**DESCRIPTION** This routine returns the segment in the list immediately following *segmentId*.

**RETURNS** A pointer to the segment ID, or NULL if there is no next segment.

**SEE ALSO** **VXWModule**

---

## VXWModule::VXWModule()

**NAME** VXWModule::VXWModule() – build module object from module ID (WFC Opt.)

**SYNOPSIS**

```
VXWModule
(
 MODULE_ID aModuleId
)
```

**DESCRIPTION** Use this constructor to manipulate a module that was not loaded using C++ interfaces. The argument *id* is the module identifier returned and used by the C interface to the VxWorks target-resident load facility.

**RETURNS** N/A.

**SEE ALSO** VXWModule, loadLib

---

## VXWModule::VXWModule()

**NAME** VXWModule::VXWModule() – load object module at memory addresses (WFC Opt.)

**SYNOPSIS**

```
VXWModule
(
 int fd,
 int symFlag,
 char * *ppText,
 char * *ppData=0,
 char * *ppBss=0
)
```

**DESCRIPTION** This constructor reads an object module from *fd*, and loads the code, data, and BSS segments at the specified load addresses in memory set aside by the caller using **VXWMemPart::alloc()**, or in the system memory partition as described below. The module is properly relocated according to the relocation commands in the file. Unresolved externals will be linked to symbols found in the system symbol table. Symbols in the module being loaded can optionally be added to the system symbol table.

**LINKING UNRESOLVED EXTERNALS**

As the module is loaded, any unresolved external references are resolved by looking up the missing symbols in the the system symbol table. If found, those references are

correctly linked to the new module. If unresolved external references cannot be found in the system symbol table, then an error message ("undefined symbol: ...") is printed for the symbol, but the loading/linking continues. In this case, NULL is returned after the module is loaded.

#### ADDING SYMBOLS TO THE SYMBOL TABLE

The symbols defined in the module to be loaded may be optionally added to the target-resident system symbol table, depending on the value of *symFlag*:

##### LOAD\_NO\_SYMBOLS

add no symbols to the system symbol table

##### LOAD\_LOCAL\_SYMBOLS

add only local symbols to the system symbol table

##### LOAD\_GLOBAL\_SYMBOLS

add only external symbols to the system symbol table

##### LOAD\_ALL\_SYMBOLS

add both local and external symbols to the system symbol table

##### HIDDEN\_MODULE

do not display the module via *moduleShow()*.

In addition, the following symbols are added to the symbol table to indicate the start of each segment: *file\_text*, *file\_data*, and *file\_bss*, where *file* is the name associated with the fd.

#### RELOCATION

The relocation commands in the object module are used to relocate the text, data, and BSS segments of the module. The location of each segment can be specified explicitly, or left unspecified in which case memory is allocated for the segment from the system memory partition. This is determined by the parameters *ppText*, *ppData*, and *ppBss*, each of which can have the following values:

##### NULL

no load address is specified, none will be returned;

##### A pointer to LD\_NO\_ADDRESS

no load address is specified, the return address is referenced by the pointer;

##### A pointer to an address

the load address is specified.

The *ppText*, *ppData*, and *ppBss* parameters specify where to load the text, data, and bss sections respectively. Each of these parameters is a pointer to a pointer; for example, *\*\*ppText* gives the address where the text segment is to begin.

For any of the three parameters, there are two ways to request that new memory be allocated, rather than specifying the section's starting address: you can either specify the parameter itself as NULL, or you can write the constant LD\_NO\_ADDRESS in place of an address. In the second case, this constructor replaces the LD\_NO\_ADDRESS value with the address actually used for each section (that is, it records the address at *\*ppText*, *\*ppData*, or *\*ppBss*).



The double indirection not only permits reporting the addresses actually used, but also allows you to specify loading a segment at the beginning of memory, since the following cases can be distinguished:

- (1) Allocate memory for a section (text in this example): `ppText == NULL`
- (2) Begin a section at address zero (the text section, below): `*ppText == 0`

Note that `loadModule()` is equivalent to this routine if all three of the segment-address parameters are set to NULL.

**COMMON**

Some host compiler/linker combinations internally use another storage class known as *common*. In the C language, uninitialized global variables are eventually put in the BSS segment. However, in partially linked object modules they are flagged internally as common and the static linker on the host resolves these and places them in BSS as a final step in creating a fully linked object module. However, the VxWorks target-resident dynamic loader is most often used to load partially linked object modules. When the VxWorks loader encounters a variable labeled as common, memory for the variable is allocated, and the variable is entered in the system symbol table (if specified) at that address. Note that most static loaders have an option that forces resolution of the common storage while leaving the module relocatable.

**RETURNS**

N/A.

**SEE ALSO**

**VXWModule**, *VxWorks Programmer's Guide: C++ Development*

## **VXWModule::VXWModule()**

**NAME**

**VXWModule::VXWModule()** – load an object module into memory (WFC Opt.)

**SYNOPSIS**

```
VXWModule
(
 int fd,
 int symFlag
)
```

**DESCRIPTION**

This constructor loads an object module from the file descriptor *fd*, and places the code, data, and BSS into memory allocated from the system memory pool.

**RETURNS**

N/A.

**SEE ALSO**

**VXWModule**

---

## **VXWModule::VXWModule()**

|                    |                                                                                                                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWModule::VXWModule()</b> – create and initialize an object module (WFC Opt.)                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>VXWModule (     char * name,     int   format,     int   flags )</pre>                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This constructor creates an object module descriptor. It is usually called from another constructor.</p> <p>The arguments specify the name of the object module file, the object module format, and a collection of options <i>flags</i>.</p> <p>Space for the new module is dynamically allocated.</p> |
| <b>RETURNS</b>     | N/A.                                                                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>VXWModule</b>                                                                                                                                                                                                                                                                                           |

---

## **VXWModule::~VXWModule()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWModule::~VXWModule()</b> – unload an object module (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>~VXWModule ()</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | <p>This destructor unloads the object module from the target system. For a.out and ECOFF format modules, unloading does the following:</p> <ol style="list-style-type: none"><li>(1) It frees the space allocated for text, data, and BSS segments, unless <b>VXWModule::VXWModule()</b> was called with specific addresses, in which case the application is responsible for freeing space.</li><li>(2) It removes all symbols associated with the object module from the system symbol table.</li><li>(3) It removes the module descriptor from the module list.</li></ol> <p>For other modules of other formats, unloading has similar effects.</p> <p>Unloading modules with this interface has no effect on breakpoints in other modules.</p> |

**RETURNS** N/A.

**SEE ALSO** VXWModule

---

## **VXWMSem::giveForce()**

**NAME** VXWMSem::giveForce() – give mutex semaphore without restrictions (WFC Opt.)

**SYNOPSIS** STATUS giveForce ()

**DESCRIPTION** This routine gives a mutual-exclusion semaphore, regardless of semaphore ownership. It is intended as a debugging aid only.

The routine is particularly useful when a task dies while holding some mutual-exclusion semaphore, because the semaphore can be resurrected. The routine gives the semaphore to the next task in the pend queue, or makes the semaphore full if no tasks are pending. In effect, execution continues as if the task owning the semaphore had actually given the semaphore.

**CAVEATS** Use this routine should only as a debugging aid, when the condition of the semaphore is known.

**RETURNS** OK.

**SEE ALSO** VXWSem::give()

---

## **VXWMSem::VXWMSem()**

**NAME** VXWMSem::VXWMSem() – create and initialize a mutex semaphore (WFC Opt.)

**SYNOPSIS** VXWMSem  
(  
int opts  
)

**DESCRIPTION** This routine allocates and initializes a mutual-exclusion semaphore. The semaphore state is initialized to full.

Semaphore options include the following:

**SEM\_Q\_PRIORITY**

Queue pending tasks on the basis of their priority.

**SEM\_Q\_FIFO**

Queue pending tasks on a first-in-first-out basis.

**SEM\_DELETE\_SAFE**

Protect a task that owns the semaphore from unexpected deletion. This option enables an implicit *taskSafe()* for each *VXWSem::take()*, and an implicit *taskUnsafe()* for each *VXWSem::give()*.

**SEM\_INVERSION\_SAFE**

Protect the system from priority inversion. With this option, the task owning the semaphore executes at the highest priority of the tasks pending on the semaphore, if that is higher than its current priority. This option must be accompanied by the **SEM\_Q\_PRIORITY** queuing mode.

Mutual-exclusion semaphores offer convenient options suited for situations that require mutually exclusive access to resources. Typical applications include sharing devices and protecting data structures. Mutual-exclusion semaphores are used by many higher-level VxWorks facilities.

The mutual-exclusion semaphore is a specialized version of the binary semaphore, designed to address issues inherent in mutual exclusion, such as recursive access to resources, priority inversion, and deletion safety. The fundamental behavior of the mutual-exclusion semaphore is identical to the binary semaphore as described for *VXWBSem::VXWBSem()*, except for the following restrictions:

- It can only be used for mutual exclusion.
- It can only be given by the task that took it.
- It may not be taken or given from interrupt level.
- The *VXWSem::flush()* operation is illegal.

These last two operations have no meaning in mutual-exclusion situations.

**RECURSIVE RESOURCE ACCESS**

A special feature of the mutual-exclusion semaphore is that it may be taken "recursively;" that is, it can be taken more than once by the task that owns it before finally being released. Recursion is useful for a set of routines that need mutually exclusive access to a resource, but may need to call each other.

Recursion is possible because the system keeps track of which task currently owns a mutual-exclusion semaphore. Before being released, a mutual-exclusion semaphore taken recursively must be given the same number of times it has been taken; this is tracked by means of a count which increments with each *VXWSem::take()* and decrements with each *VXWSem::give()*.

#### **PRIORITY-INVERSION SAFETY**

If the option `SEM_INVERSION_SAFE` is selected, the library adopts a priority-inheritance protocol to resolve potential occurrences of "priority inversion," a problem stemming from the use semaphores for mutual exclusion. Priority inversion arises when a higher-priority task is forced to wait an indefinite period of time for the completion of a lower-priority task.

Consider the following scenario: T1, T2, and T3 are tasks of high, medium, and low priority, respectively. T3 has acquired some resource by taking its associated semaphore. When T1 preempts T3 and contends for the resource by taking the same semaphore, it becomes blocked. If we could be assured that T1 would be blocked no longer than the time it normally takes T3 to finish with the resource, the situation would not be problematic. However, the low-priority task is vulnerable to preemption by medium-priority tasks; a preempting task, T2, could inhibit T3 from relinquishing the resource. This condition could persist, blocking T1 for an indefinite period of time.

The priority-inheritance protocol solves the problem of priority inversion by elevating the priority of T3 to the priority of T1 during the time T1 is blocked on T3. This protects T3, and indirectly T1, from preemption by T2. Stated more generally, the priority-inheritance protocol assures that a task which owns a resource executes at the priority of the highest priority task blocked on that resource. When execution is complete, the task gives up the resource and returns to its normal, or standard, priority. Hence, the "inheriting" task is protected from preemption by any intermediate-priority tasks.

The priority-inheritance protocol also takes into consideration a task's ownership of more than one mutual-exclusion semaphore at a time. Such a task will execute at the priority of the highest priority task blocked on any of the resources it owns. The task returns to its normal priority only after relinquishing all of its mutual-exclusion semaphores that have the inversion-safety option enabled.

#### **SEMAPHORE DELETION**

The `VXWSem::~~VXWSem()` destructor terminates a semaphore and deallocates any associated memory. The deletion of a semaphore unblocks tasks pended on that semaphore; the routines which were pended return `ERROR`. Take special care when deleting mutual-exclusion semaphores to avoid deleting a semaphore out from under a task that already owns (has taken) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task owns.

#### **TASK-DELETION SAFETY**

If the option `SEM_DELETE_SAFE` is selected, the task owning the semaphore is protected from deletion as long as it owns the semaphore. This solves another problem endemic to mutual exclusion. Deleting a task executing in a critical region can be catastrophic. The resource could be left in a corrupted state and the semaphore guarding the resource would be unavailable, effectively shutting off all access to the resource.

As discussed in `taskLib`, the primitives `taskSafe()` and `taskUnsafe()` offer one solution, but as this type of protection goes hand in hand with mutual exclusion, the

**VXWMsgQ::info()**

mutual-exclusion semaphore provides the option `SEM_DELETE_SAFE`, which enables an implicit `taskSafe()` with each `VXWSem::take()`, and a `taskUnsafe()` with each `VXWSem::give()`. This convenience is also more efficient, as the resulting code requires fewer entrances to the kernel.

- CAVEATS** There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible. Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores will not be given back, effectively leaving a resource permanently unavailable. The `SEM_DELETE_SAFE` option partially protects an application, to the extent that unexpected deletions will be deferred until the resource is released.
- RETURNS** N/A
- SEE ALSO** `VXWSem`, `taskSafe()`, `taskUnsafe()`

---

## VXWMsgQ::info()

**NAME** `VXWMsgQ::info()` – get information about message queue (WFC Opt.)

**SYNOPSIS**

```
STATUS info
(
 MSG_Q_INFO * pInfo
) const
```

**DESCRIPTION** This routine gets information about the state and contents of its message queue. The parameter `pInfo` is a pointer to a structure of type `MSG_Q_INFO` defined in `msgQLib.h` as follows:

```
typedef struct /* MSG_Q_INFO */
{
 int numMsgs; /* OUT: number of messages queued */
 int numTasks; /* OUT: number of tasks waiting on msg q */
 int sendTimeouts; /* OUT: count of send timeouts */
 int rcvTimeouts; /* OUT: count of receive timeouts */
 int options; /* OUT: options with which msg q was created */
 int maxMsgs; /* OUT: max messages that can be queued */
 int maxMsgLength; /* OUT: max byte length of each message */
 int taskIdListMax; /* IN: max tasks to fill in taskIdList */
 int * taskIdList; /* PTR: array of task IDs waiting on msg q */
 int msgListMax; /* IN: max msgs to fill in msg lists */
}
```

```

char ** msgPtrList; /* PTR: array of msg ptrs queued to msg q */
int * msgLenList; /* PTR: array of lengths of msgs */
} MSG_Q_INFO;

```

If the message queue is empty, there may be tasks blocked on receiving. If the message queue is full, there may be tasks blocked on sending. This can be determined as follows:

- If *numMsgs* is 0, then *numTasks* indicates the number of tasks blocked on receiving.
- If *numMsgs* is equal to *maxMsgs*, then *numTasks* is the number of tasks blocked on sending.
- If *numMsgs* is greater than 0 but less than *maxMsgs*, then *numTasks* will be 0.

A list of pointers to the messages queued and their lengths can be obtained by setting *msgPtrList* and *msgLenList* to the addresses of arrays to receive the respective lists, and setting *msgListMax* to the maximum number of elements in those arrays. If either list pointer is NULL, no data is returned for that array.

No more than *msgListMax* message pointers and lengths are returned, although *numMsgs* is always returned with the actual number of messages queued.

For example, if the caller supplies a *msgPtrList* and *msgLenList* with room for 10 messages and sets *msgListMax* to 10, but there are 20 messages queued, then the pointers and lengths of the first 10 messages in the queue are returned in *msgPtrList* and *msgLenList*, but *numMsgs* is returned with the value 20.

A list of the task IDs of tasks blocked on the message queue can be obtained by setting *taskIdList* to the address of an array to receive the list, and setting *taskIdListMax* to the maximum number of elements in that array. If *taskIdList* is NULL, then no task IDs are returned. No more than *taskIdListMax* task IDs are returned, although *numTasks* is always returned with the actual number of tasks blocked.

For example, if the caller supplies a *taskIdList* with room for 10 task IDs and sets *taskIdListMax* to 10, but there are 20 tasks blocked on the message queue, then the IDs of the first 10 tasks in the blocked queue are returned in *taskIdList*, but *numTasks* is returned with the value 20.

Note that the tasks returned in *taskIdList* may be blocked for either send or receive. As noted above this can be determined by examining *numMsgs*. The variables *sendTimeouts* and *recvTimeouts* are the counts of the number of times *VXWMsgQ::send()* and *VXWMsgQ::receive()* (or their equivalents in other language bindings) respectively returned with a timeout.

The variables *options*, *maxMsgs*, and *maxMsgLength* are the parameters with which the message queue was created.

**WARNING**

The information returned by this routine is not static and may be obsolete by the time it is examined. In particular, the lists of task IDs and/or message pointers may no longer be valid. However, the information is obtained atomically, thus it is an accurate snapshot of

## ***VXWMsgQ::numMsgs()***

the state of the message queue at the time of the call. This information is generally used for debugging purposes only.

- WARNING** The current implementation of this routine locks out interrupts while obtaining the information. This can compromise the overall interrupt latency of the system. Generally this routine is used for debugging purposes only.
- RETURNS** OK or ERROR.
- SEE ALSO** **VXWMsgQ**

---

## ***VXWMsgQ::numMsgs()***

- NAME** ***VXWMsgQ::numMsgs()*** – report the number of messages queued (WFC Opt.)
- SYNOPSIS** `int numMsgs ()`
- DESCRIPTION** This routine returns the number of messages currently queued to the message queue.
- RETURNS** The number of messages queued, or ERROR.
- ERRNO** **S\_objLib\_OBJ\_ID\_ERROR**  
– *msgQId* is invalid.
- SEE ALSO** **VXWMsgQ**

---

## ***VXWMsgQ::receive()***

- NAME** ***VXWMsgQ::receive()*** – receive a message from message queue (WFC Opt.)
- SYNOPSIS** `int receive  
(  
  char * buffer,  
  UINT nBytes,  
  int timeout  
)`



**DESCRIPTION** This routine receives a message from its message queue. The received message is copied into the specified *buffer*, which is *nBytes* in length. If the message is longer than *nBytes*, the remainder of the message is discarded (no error indication is returned).

The *timeout* parameter specifies the number of ticks to wait for a message to be sent to the queue, if no message is available when **VXWMsgQ::receive()** is called. The *timeout* parameter can also have the following special values:

**NO\_WAIT**  
return immediately, even if the message has not been sent.

**WAIT\_FOREVER**  
never time out.

**WARNING** This routine must not be called by interrupt service routines.

**RETURNS** The number of bytes copied to *buffer*, or ERROR.

**ERRNO**

**S\_objLib\_OBJ\_DELETED**  
– the message queue was deleted while waiting to receive a message.

**S\_objLib\_OBJ\_UNAVAILABLE**  
– *timeout* is set to **NO\_WAIT**, and no messages are available.

**S\_objLib\_OBJ\_TIMEOUT**  
– no messages were received in *timeout* ticks.

**S\_msgQLib\_INVALID\_MSG\_LENGTH**  
– *nBytes* is less than 0.

**SEE ALSO** **VXWMsgQ**

---

## VXWMsgQ::send()

**NAME** **VXWMsgQ::send()** – send a message to message queue (WFC Opt.)

**SYNOPSIS**

```
STATUS send
(
 char * buffer,
 UINT nBytes,
 int timeout,
 int pri
)
```

**DESCRIPTION** This routine sends the message in *buffer* of length *nBytes* to its message queue. If any tasks are already waiting to receive messages on the queue, the message is immediately

delivered to the first waiting task. If no task is waiting to receive messages, the message is saved in the message queue.

The *timeout* parameter specifies the number of ticks to wait for free space if the message queue is full. The *timeout* parameter can also have the following special values:

**NO\_WAIT**

return immediately, even if the message has not been sent.

**WAIT\_FOREVER**

never time out.

The *pri* parameter specifies the priority of the message being sent. The possible values are:

**MSG\_PRI\_NORMAL**

normal priority; add the message to the tail of the list of queued messages.

**MSG\_PRI\_URGENT**

urgent priority; add the message to the head of the list of queued messages.

**USE BY INTERRUPT SERVICE ROUTINES**

This routine can be called by interrupt service routines as well as by tasks. This is one of the primary means of communication between an interrupt service routine and a task. When called from an interrupt service routine, *timeout* must be **NO\_WAIT**.

**RETURNS**

OK or ERROR.

**ERRNO**

**S\_objLib\_OBJ\_DELETED**

– the message queue was deleted while waiting to a send message.

**S\_objLib\_OBJ\_UNAVAILABLE**

– *timeout* is set to **NO\_WAIT**, and the queue is full.

**S\_objLib\_OBJ\_TIMEOUT**

– the queue is full for *timeout* ticks.

**S\_msgQLib\_INVALID\_MSG\_LENGTH**

– *nBytes* is larger than the *maxMsgLength* set for the message queue.

**S\_msgQLib\_NON\_ZERO\_TIMEOUT\_AT\_INT\_LEVEL**

– called from an ISR, with *timeout* not set to **NO\_WAIT**.

**SEE ALSO**

**VXWMsgQ**

## VXWMsgQ::show()

**NAME** VXWMsgQ::show() – show information about a message queue (WFC Opt.)

**SYNOPSIS**

```
STATUS show
(
 int level
) const
```

**DESCRIPTION** This routine displays the state and optionally the contents of a message queue.

A summary of the state of the message queue is displayed as follows:

```
Message Queue Id : 0x3f8c20
Task Queuing : FIFO
Message Byte Len : 150
Messages Max : 50
Messages Queued : 0
Receivers Blocked : 1
Send timeouts : 0
Receive timeouts : 0
```

If *level* is 1, more detailed information is displayed. If messages are queued, they are displayed as follows:

```
Messages queued:
address length value
1 0x123eb204 4 0x00000001 0x12345678
```

If tasks are blocked on the queue, they are displayed as follows:

```
Receivers blocked:
NAME TID PRI DELAY

tExcTask 3fd678 0 21
```

**RETURNS** OK or ERROR.

**SEE ALSO** VXWMsgQ

---

## VXWMsgQ::VXWMsgQ()

|                    |                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | VXWMsgQ::VXWMsgQ() – create and initialize a message queue (WFC Opt.)                                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>VXWMsgQ (     int maxMsgs,     int maxMsgLen,     int opts )</pre>                                                                                                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | <p>This constructor creates a message queue capable of holding up to <i>maxMsgs</i> messages, each up to <i>maxMsgLen</i> bytes long. The queue can be created with the following options specified as <i>opts</i>:</p> <p><b>MSG_Q_FIFO</b><br/>queue pended tasks in FIFO order.</p> <p><b>MSG_Q_PRIORITY</b><br/>queue pended tasks in priority order.</p> |
| <b>RETURNS</b>     | N/A.                                                                                                                                                                                                                                                                                                                                                          |
| <b>ERRNO</b>       | <p><b>S_memLib_NOT_ENOUGH_MEMORY</b><br/>– unable to allocate memory for message queue and message buffers.</p> <p><b>S_intLib_NOT_ISR_CALLABLE</b><br/>– called from an interrupt service routine.</p>                                                                                                                                                       |
| <b>SEE ALSO</b>    | VXWMsgQ, vxwSmLib                                                                                                                                                                                                                                                                                                                                             |

---

## VXWMsgQ::VXWMsgQ()

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <b>NAME</b>     | VXWMsgQ::VXWMsgQ() – build message-queue object from ID (WFC Opt.) |
| <b>SYNOPSIS</b> | <pre>VXWMsgQ (     MSG_Q_ID id )</pre>                             |

**DESCRIPTION** Use this constructor to manipulate a message queue that was not created using C++ interfaces. The argument *id* is the message-queue identifier returned and used by the C interface to the VxWorks message queue facility.

**RETURNS** N/A.

**SEE ALSO** **VXWMsgQ**, **msgQLib**

## ***VXWMsgQ::~VXWMsgQ()***

**NAME** ***VXWMsgQ::~VXWMsgQ()*** – delete message queue (WFC Opt.)

**SYNOPSIS** `virtual ~VXWMsgQ ()`

**DESCRIPTION** This destructor deletes a message queue. Any task blocked on either a ***VXWMsgQ::send()*** or ***VXWMsgQ::receive()*** is unblocked and receives an error from the call with **errno** set to **S\_objLib\_OBJECT\_DELETED**.

**RETURNS** N/A.

**ERRNO** **S\_objLib\_OBJ\_ID\_ERROR**  
 – *msgQId* is invalid.

**S\_intLib\_NOT\_ISR\_CALLABLE**  
 – called from an interrupt service routine.

**SEE ALSO** **VXWMsgQ**

## ***VXWRingBuf::flush()***

**NAME** ***VXWRingBuf::flush()*** – make ring buffer empty (WFC Opt.)

**SYNOPSIS** `void flush ()`

**DESCRIPTION** This routine initializes the ring buffer to be empty. Any data in the buffer is lost.

**RETURNS** N/A

**SEE ALSO** **VXWRingBuf**

---

## VXWRingBuf::freeBytes()

**NAME** `VXWRingBuf::freeBytes()` – determine the number of free bytes in ring buffer (WFC Opt.)

**SYNOPSIS**

```
int freeBytes ()
```

**DESCRIPTION** This routine determines the number of bytes currently unused in the ring buffer.

**RETURNS** The number of unused bytes in the ring buffer.

**SEE ALSO** `VXWRingBuf`

---

## VXWRingBuf::get()

**NAME** `VXWRingBuf::get()` – get characters from ring buffer (WFC Opt.)

**SYNOPSIS**

```
int get
(
 char * buffer,
 int maxbytes
)
```

**DESCRIPTION** This routine copies bytes from the ring buffer into *buffer*. It copies as many bytes as are available in the ring, up to *maxbytes*. The bytes copied are then removed from the ring.

**RETURNS** The number of bytes actually received from the ring buffer; it may be zero if the ring buffer is empty at the time of the call.

**SEE ALSO** `VXWRingBuf`

---

## VXWRingBuf::isEmpty()

**NAME** `VXWRingBuf::isEmpty()` – test whether ring buffer is empty (WFC Opt.)

**SYNOPSIS**

```
BOOL isEmpty ()
```

**DESCRIPTION** This routine reports on whether the ring buffer is empty.

**RETURNS** TRUE if empty, FALSE if not.

**SEE ALSO** VXWRingBuf

---

## ***VXWRingBuf::isFull()***

**NAME** *VXWRingBuf::isFull()* – test whether ring buffer is full (no more room) (WFC Opt.)

**SYNOPSIS** `BOOL isFull ()`

**DESCRIPTION** This routine reports on whether the ring buffer is completely full.

**RETURNS** TRUE if full, FALSE if not.

**SEE ALSO** VXWRingBuf

---

## ***VXWRingBuf::moveAhead()***

**NAME** *VXWRingBuf::moveAhead()* – advance ring pointer by *n* bytes (WFC Opt.)

**SYNOPSIS** `void moveAhead  
(  
int n  
)`

**DESCRIPTION** This routine advances the ring buffer input pointer by *n* bytes. This makes *n* bytes available in the ring buffer, after having been written ahead in the ring buffer with *VXWRingBuf::putAhead()*.

**RETURNS** N/A

**SEE ALSO** VXWRingBuf

## **VXWRingBuf::nBytes()**

**NAME** *VXWRingBuf::nBytes()* – determine the number of bytes in ring buffer (WFC Opt.)

**SYNOPSIS** `int nBytes ()`

**DESCRIPTION** This routine determines the number of bytes currently in the ring buffer.

**RETURNS** The number of bytes filled in the ring buffer.

**SEE ALSO** **VXWRingBuf**

---

## **VXWRingBuf::put()**

**NAME** *VXWRingBuf::put()* – put bytes into ring buffer (WFC Opt.)

**SYNOPSIS** `int put  
(  
    char * buffer,  
    int nBytes  
)`

**DESCRIPTION** This routine puts bytes from *buffer* into the ring buffer. The specified number of bytes is put into the ring, up to the number of bytes available in the ring.

**RETURNS** The number of bytes actually put into the ring buffer; it may be less than number requested, even zero, if there is insufficient room in the ring buffer at the time of the call.

**SEE ALSO** **VXWRingBuf**



---

## VXWRingBuf::putAhead()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWRingBuf::putAhead()</b> – put a byte ahead in a ring buffer without moving ring pointers (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>void putAhead (     char byte,     int offset )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>DESCRIPTION</b> | <p>This routine writes a byte into the ring, but does not move the ring buffer pointers. Thus the byte is not yet be available to <b>VXWRingBuf::get()</b> calls. The byte is written <i>offset</i> bytes ahead of the next input location in the ring. Thus, an offset of 0 puts the byte in the same position as <b>VXWRingBuf::put()</b> would put a byte, except that the input pointer is not updated.</p> <p>Bytes written ahead in the ring buffer with this routine can be made available all at once by subsequently moving the ring buffer pointers with the routine <b>VXWRingBuf::moveAhead()</b>.</p> <p>Before calling <b>VXWRingBuf::putAhead()</b>, the caller must verify that at least <i>offset</i> + 1 bytes are available in the ring buffer.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <b>VXWRingBuf</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

---

## VXWRingBuf::VXWRingBuf()

|                    |                                                                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWRingBuf::VXWRingBuf()</b> – create an empty ring buffer (WFC Opt.)                                                                                      |
| <b>SYNOPSIS</b>    | <pre>VXWRingBuf (     int nbytes )</pre>                                                                                                                      |
| <b>DESCRIPTION</b> | <p>This constructor creates a ring buffer of size <i>nbytes</i>, and initializes it. Memory for the buffer is allocated from the system memory partition.</p> |
| <b>RETURNS</b>     | N/A.                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>VXWRingBuf</b>                                                                                                                                             |

---

## **VXWRingBuf::VXWRingBuf( )**

|                    |                                                                                                                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWRingBuf::VXWRingBuf( )</b> – build ring-buffer object from existing ID (WFC Opt.)                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>VXWRingBuf (     RING_ID aRingId )</pre>                                                                                                                                                                 |
| <b>DESCRIPTION</b> | Use this constructor to build a ring-buffer object from an existing ring buffer. This permits you to use the C++ ring-buffer interfaces even if the ring buffer itself was created by a routine written in C. |
| <b>RETURNS</b>     | N/A.                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>VXWRingBuf</b> , <b>rngLib</b>                                                                                                                                                                             |

---

## **VXWRingBuf::~VXWRingBuf( )**

|                    |                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWRingBuf::~VXWRingBuf( )</b> – delete ring buffer (WFC Opt.)                                          |
| <b>SYNOPSIS</b>    | <pre>~VXWRingBuf ( )</pre>                                                                                 |
| <b>DESCRIPTION</b> | This destructor deletes a specified ring buffer. Any data in the buffer at the time it is deleted is lost. |
| <b>RETURNS</b>     | N/A                                                                                                        |
| <b>SEE ALSO</b>    | <b>VXWRingBuf</b>                                                                                          |

---

## **VXWSem::flush( )**

|                 |                                                                               |
|-----------------|-------------------------------------------------------------------------------|
| <b>NAME</b>     | <b>VXWSem::flush( )</b> – unblock every task pended on a semaphore (WFC Opt.) |
| <b>SYNOPSIS</b> | <pre>STATUS flush ( )</pre>                                                   |

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine atomically unblocks all tasks pended on a specified semaphore; that is, all tasks are unblocked before any is allowed to run. The state of the underlying semaphore is unchanged. All pended tasks enter the ready queue before having a chance to execute. The flush operation is useful as a means of broadcast in synchronization applications. Its use is illegal for mutual-exclusion semaphores created with <i>VXWMSem::VXWMSem()</i> . |
| <b>RETURNS</b>     | OK, or ERROR if the operation is not supported.                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <i>VXWSem</i> , <i>VXWCsem::VXWCsem()</i> , <i>VXWBSem::VXWBsem()</i> , <i>VXWMSem::VXWMsem()</i> , <i>VxWorks Programmer's Guide: Basic OS</i>                                                                                                                                                                                                                                                                                                             |

---

## **VXWSem::give()**

|                    |                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>VXWSem::give()</i> – give a semaphore (WFC Opt.)                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <code>STATUS give ()</code>                                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | This routine performs the give operation on a specified semaphore. Depending on the type of semaphore, the state of the semaphore and of the pending tasks may be affected. The behavior of <i>VXWSem::give()</i> is discussed fully in the constructor description for the specific semaphore type being used. |
| <b>RETURNS</b>     | OK.                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <i>VXWSem</i> , <i>VXWCsem::VXWCsem()</i> , <i>VXWBSem::VXWBsem()</i> , <i>VXWMSem::VXWMsem()</i> , <i>VxWorks Programmer's Guide: Basic OS</i>                                                                                                                                                                 |

---

## **VXWSem::id()**

|                    |                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>VXWSem::id()</i> – reveal underlying semaphore ID (WFC Opt.)                                                                               |
| <b>SYNOPSIS</b>    | <code>SEM_ID id ()</code>                                                                                                                     |
| <b>DESCRIPTION</b> | This routine returns the semaphore ID corresponding to a semaphore object. The semaphore ID is used by the C interface to VxWorks semaphores. |
| <b>RETURNS</b>     | Semaphore ID.                                                                                                                                 |
| <b>SEE ALSO</b>    | <i>VXWSem</i> , <i>semLib</i>                                                                                                                 |

**VXWSem::info()**

---

**VXWSem::info()**

|                    |                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWSem::info()</b> – get a list of task IDs that are blocked on a semaphore (WFC Opt.)                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS info (     int idList[],     int maxTasks ) const</pre>                                                                                                          |
| <b>DESCRIPTION</b> | This routine reports the tasks blocked on a specified semaphore. Up to <i>maxTasks</i> task IDs are copied to the array specified by <i>idList</i> . The array is unordered. |
| <b>WARNING</b>     | There is no guarantee that all listed tasks are still valid or that new tasks have not been blocked by the time <b>VXWSem::info()</b> returns.                               |
| <b>RETURNS</b>     | The number of blocked tasks placed in <i>idList</i> .                                                                                                                        |
| <b>SEE ALSO</b>    | <b>VXWSem</b>                                                                                                                                                                |

---

**VXWSem::show()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWSem::show()</b> – show information about a semaphore (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>STATUS show (     int level ) const</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This routine displays (on standard output) the state and optionally the pended tasks of a semaphore.</p> <p>A summary of the state of the semaphore is displayed as follows:</p> <pre>Semaphore Id      : 0x585f2 Semaphore Type    : BINARY Task Queuing      : PRIORITY Pended Tasks      : 1 State             : EMPTY {Count if COUNTING, Owner if MUTEX}</pre> <p>If <i>level</i> is 1, more detailed information is displayed. If tasks are blocked on the queue, they are displayed in the order in which they will unblock, as follows:</p> |

| NAME     | TID    | PRI | DELAY |
|----------|--------|-----|-------|
| tExcTask | 3fd678 | 0   | 21    |
| tLogTask | 3f8ac0 | 0   | 611   |

**RETURNS** OK or ERROR.

**SEE ALSO** VXWSem

## VXWSem::take()

**NAME** VXWSem::take() – take a semaphore (WFC Opt.)

**SYNOPSIS**

```
STATUS take
(
 int timeout
)
```

**DESCRIPTION** This routine performs the take operation on a specified semaphore. Depending on the type of semaphore, the state of the semaphore and the calling task may be affected. The behavior of *VXWSem::take()* is discussed fully in the constructor description for the specific semaphore type being used.

A timeout in ticks may be specified. If a task times out, *VXWSem::take()* returns ERROR. Timeouts of **WAIT\_FOREVER** and **NO\_WAIT** indicate to wait indefinitely or not to wait at all.

When *VXWSem::take()* returns due to timeout, it sets the errno to **S\_objLib\_OBJ\_TIMEOUT** (defined in **objLib.h**).

The *VXWSem::take()* routine must not be called from interrupt service routines.

**RETURNS** OK, or ERROR if the task timed out.

**SEE ALSO** VXWSem, VXWCSem::VXWCsem(), VXWBSem::VXWBsem(), VXWMSem::VXWMsem(), *VxWorks Programmer's Guide: Basic OS*

## **VXWSem::VXWSem()**

|                    |                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWSem::VXWSem()</b> – build semaphore object from semaphore ID (WFC Opt.)                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>VXWSem<br/>(<br/>    SEM_ID id<br/>)</pre>                                                                                                                                                                      |
| <b>DESCRIPTION</b> | Use this constructor to manipulate a semaphore that was not created using C++ interfaces. The argument <i>id</i> is the semaphore identifier returned and used by the C interface to the VxWorks semaphore facility. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>VXWSem</b> , <b>semLib</b>                                                                                                                                                                                        |

---

## **VXWSem::~VXWSem()**

|                    |                                                                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWSem::~VXWSem()</b> – delete a semaphore (WFC Opt.)                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>virtual ~VXWSem ()</pre>                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | This destructor terminates and deallocates any memory associated with a specified semaphore. Any pending tasks unblock and return ERROR.                                                                                                                                                              |
| <b>WARNING</b>     | Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that already has taken (owns) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully taken. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <b>VXWSem</b> , <i>VxWorks Programmer's Guide: Basic OS</i>                                                                                                                                                                                                                                           |

---

## VXWsmName::nameGet( )

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>VXWsmName::nameGet( )</b> – get name and type of a shared memory object (VxMP Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>     | <pre>STATUS nameGet (     char * name,     int * pType,     int waitType )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>  | <p>This routine searches the shared memory name database for an object matching this VXWsmName instance. If the object is found, its name and type are copied to the addresses pointed to by <i>name</i> and <i>pType</i>. The value of <i>waitType</i> can be one of the following:</p> <p><b>NO_WAIT (0)</b><br/>The call returns immediately, even if the object value is not in the database</p> <p><b>WAIT_FOREVER (-1)</b><br/>The call returns only when the object value is available in the database.</p> |
| <b>AVAILABILITY</b> | This routine depends on the unbundled shared memory objects support option, VxMP.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>RETURNS</b>      | OK, or ERROR if <i>value</i> is not found or if the wait type is invalid.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>ERRNO</b>        | <b>S_smNameLib_NOT_INITIALIZED</b><br><b>S_smNameLib_VALUE_NOT_FOUND</b><br><b>S_smNameLib_INVALID_WAIT_TYPE</b><br><b>S_smObjLib_LOCK_TIMEOUT</b>                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>     | <b>VXWsmName</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

---

## VXWsmName::nameGet( )

|                 |                                                                                          |
|-----------------|------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <b>VXWsmName::nameGet( )</b> – get name of a shared memory object (VxMP Opt.) (WFC Opt.) |
| <b>SYNOPSIS</b> | <pre>STATUS nameGet (     char * name,     int waitType )</pre>                          |

**VXWSmName::nameSet()**

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b>  | This routine searches the shared memory name database for an object matching this VXWSmName instance. If the object is found, its name is copied to the address pointed to by <i>name</i> . The value of <i>waitType</i> can be one of the following:<br><b>NO_WAIT (0)</b><br>The call returns immediately, even if the object value is not in the database<br><b>WAIT_FOREVER (-1)</b><br>The call returns only when the object value is available in the database. |
| <b>AVAILABILITY</b> | This routine depends on the unbundled shared memory objects support option, VxMP.                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>RETURNS</b>      | OK, or ERROR if <i>value</i> is not found or if the wait type is invalid.                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>ERRNO</b>        | S_smNameLib_NOT_INITIALIZED<br>S_smNameLib_VALUE_NOT_FOUND<br>S_smNameLib_INVALID_WAIT_TYPE<br>S_smObjLib_LOCK_TIMEOUT                                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>     | VXWSmName                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

---

## VXWSmName::nameSet()

|                     |                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | VXWSmName::nameSet() – define a name string in the shared-memory name database (VxMP Opt.) (WFC Opt.)                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>     | <pre>virtual STATUS nameSet (     char * name ) = 0</pre>                                                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b>  | <p>This routine adds a name of the type appropriate for each derived class to the database of memory object names.</p> <p>The <i>name</i> parameter is an arbitrary null-terminated string with a maximum of 20 characters, including EOS.</p> <p>A name can be entered only once in the database, but there can be more than one name associated with an object ID.</p> |
| <b>AVAILABILITY</b> | This routine depends on the unbundled shared memory objects support option, VxMP.                                                                                                                                                                                                                                                                                        |
| <b>RETURNS</b>      | OK, or ERROR if there is insufficient memory for <i>name</i> to be allocated, if <i>name</i> is already in the database, or if the database is already full.                                                                                                                                                                                                             |



**ERRNO**            *S\_smNameLib\_NOT\_INITIALIZED*  
*S\_smNameLib\_NAME\_TOO\_LONG*  
*S\_smNameLib\_NAME\_ALREADY\_EXIST*  
*S\_smNameLib\_DATABASE\_FULL*  
*S\_smObjLib\_LOCK\_TIMEOUT*

**SEE ALSO**        *VXWSmName*

## ***VXWSmName::~~VXWSmName()***

**NAME**            *VXWSmName::~~VXWSmName()* – remove an object from the shared memory objects name database (VxMP Opt.) (WFC Opt.)

**SYNOPSIS**        `virtual ~VXWSmName ()`

**DESCRIPTION**    This routine removes an object from the shared memory objects name database.

**AVAILABILITY**    This routine depends on code distributed as a component of the unbundled shared memory objects support option, VxMP.

**RETURNS**        OK, or ERROR if the database is not initialized, or the name-database lock times out.

**ERRNO**            *S\_smNameLib\_NOT\_INITIALIZED*  
*S\_smObjLib\_LOCK\_TIMEOUT*

**SEE ALSO**        *VXWSmName*

## ***VXWSymTab::add()***

**NAME**            *VXWSymTab::add()* – create and add a symbol to a symbol table, including a group number (WFC Opt.)

**SYNOPSIS**        `STATUS add`  
`(`  
`char *    name,`  
`char *    value,`  
`SYM_TYPE type,`  
`UINT16    group`  
`)`

**VXWSymTab::each()**

- DESCRIPTION** This routine allocates a symbol *name* and adds it to its symbol table with the specified parameters *value*, *type*, and *group*. The *group* parameter specifies the group number assigned to a module when it is loaded on the target; see the manual entry for **moduleLib**.
- RETURNS** OK, or ERROR if there is insufficient memory for the symbol to be allocated.
- SEE ALSO** **VXWSymTab**, **moduleLib**

---

## VXWSymTab::each()

**NAME** **VXWSymTab::each()** – call a routine to examine each entry in a symbol table (WFC Opt.)

**SYNOPSIS**

```

SYMBOL * each
(
 FUNCPTR routine,
 int routineArg
)

```

**DESCRIPTION** This routine calls a user-supplied routine to examine each entry in the symbol table; it calls the specified routine once for each entry. The routine must have the following type signature:

```

BOOL routine
(
 char * name, /* entry name */
 int val, /* value associated with entry */
 SYM_TYPE type, /* entry type */
 int arg, /* arbitrary user-supplied arg */
 UINT16 group /* group number */
)

```

The user-supplied routine must return TRUE if **VXWSymTab::each()** is to continue calling it for each entry, or FALSE if it is done and **VXWSymTab::each()** can exit.

- RETURNS** A pointer to the last symbol reached, or NULL if all symbols are reached.
- SEE ALSO** **VXWSymTab**

---

## VXWSymTab::findByName()

**NAME** VXWSymTab::findByName() – look up a symbol by name (WFC Opt.)

**SYNOPSIS**

```
STATUS findByName
(
 char * name,
 char * *pValue,
 SYM_TYPE * pType
) const
```

**DESCRIPTION** This routine searches its symbol table for a symbol matching a specified name. If the symbol is found, its value and type are copied to *pValue* and *pType*. If multiple symbols have the same name but differ in type, the routine chooses the matching symbol most recently added to the symbol table.

**RETURNS** OK, or ERROR if the symbol cannot be found.

**SEE ALSO** VXWSymTab

---

## VXWSymTab::findByNameAndType()

**NAME** VXWSymTab::findByNameAndType() – look up a symbol by name and type (WFC Opt.)

**SYNOPSIS**

```
STATUS findByNameAndType
(
 char * name,
 char * *pValue,
 SYM_TYPE * pType,
 SYM_TYPE goalType,
 SYM_TYPE mask
) const
```

**DESCRIPTION** This routine searches its symbol table for a symbol matching both name and type (*name* and *goalType*). If the symbol is found, its value and type are copied to *pValue* and *pType*. The *mask* parameter can be used to match sub-classes of type.

**RETURNS** OK, or ERROR if the symbol is not found.

**SEE ALSO** VXWSymTab

## **VXWSymTab::findByValue( )**

**NAME** VXWSymTab::findByValue( ) – look up a symbol by value (WFC Opt.)

**SYNOPSIS**

```
STATUS findByValue
(
 UINT value,
 char * name,
 int * pValue,
 SYM_TYPE * pType
) const
```

**DESCRIPTION** This routine searches its symbol table for a symbol matching a specified value. If there is no matching entry, it chooses the table entry with the next lower value. The symbol name (with terminating EOS), the actual value, and the type are copied to *name*, *pValue*, and *pType*.

**RETURNS** OK, or ERROR if *value* is less than the lowest value in the table.

**SEE ALSO** VXWSymTab

---

## **VXWSymTab::findByValueAndType( )**

**NAME** VXWSymTab::findByValueAndType( ) – look up a symbol by value and type (WFC Opt.)

**SYNOPSIS**

```
STATUS findByValueAndType
(
 UINT value,
 char * name,
 int * pValue,
 SYM_TYPE * pType,
 SYM_TYPE goalType,
 SYM_TYPE mask
) const
```

**DESCRIPTION** This routine searches a symbol table for a symbol matching both value and type (*value* and *goalType*). If there is no matching entry, it chooses the table entry with the next lower value. The symbol name (with terminating EOS), the actual value, and the type are copied to *name*, *pValue*, and *pType*. The *mask* parameter can be used to match sub-classes of type.

**RETURNS** OK, or ERROR if *value* is less than the lowest value in the table.

**SEE ALSO** VXWSymTab

---

## *VXWSymTab::remove()*

**NAME** *VXWSymTab::remove()* – remove a symbol from a symbol table (WFC Opt.)

**SYNOPSIS**

```
STATUS remove
(
 char * name,
 SYM_TYPE type
)
```

**DESCRIPTION** This routine removes a symbol of matching name and type from its symbol table. The symbol is deallocated if found. Note that VxWorks symbols in a standalone VxWorks image (where the symbol table is linked in) cannot be removed.

**RETURNS** OK, or ERROR if the symbol is not found or could not be deallocated.

**SEE ALSO** VXWSymTab

---

## *VXWSymTab::VXWSymTab()*

**NAME** *VXWSymTab::VXWSymTab()* – create a symbol table (WFC Opt.)

**SYNOPSIS**

```
VXWSymTab
(
 int hashSizeLog2,
 BOOL sameNameOk,
 PART_ID symPartId
)
```

**DESCRIPTION** This constructor creates and initializes a symbol table with a hash table of a specified size. The size of the hash table is specified as a power of two. For example, if *hashSizeLog2* is 6, a 64-entry hash table is created.

If *sameNameOk* is FALSE, attempting to add a symbol with the same name and type as an already-existing symbol results in an error.

**VXWSymTab::VXWSymTab( )**

Memory for storing symbols as they are added to the symbol table will be allocated from the memory partition *symPartId*. The ID of the system memory partition is stored in the global variable **memSysPartId**, which is declared in **memLib.h**.

**RETURNS** N/A

**SEE ALSO** VXWSymTab

---

## **VXWSymTab::VXWSymTab( )**

**NAME** *VXWSymTab::VXWSymTab( )* – create a symbol-table object (WFC Opt.)

**SYNOPSIS**

```
VXWSymTab
(
 SYMTAB_ID aSymTabId
)
```

**DESCRIPTION** This constructor creates a symbol table object based on an existing symbol table. For example, the following statement creates a symbol-table object for the VxWorks system symbol table (assuming you have configured a target-resident symbol table into your VxWorks system):

```
VXWSymTab sSym;
...
sSym = VXWSymTab (sysSymTbl);
```

**SEE ALSO** VXWSymTab

---

## **VXWSymTab::~VXWSymTab( )**

**NAME** *VXWSymTab::~VXWSymTab( )* – delete a symbol table (WFC Opt.)

**SYNOPSIS**

```
~VXWSymTab ()
```

**DESCRIPTION** This routine deletes a symbol table; it deallocates all memory associated with its symbol table, including the hash table, and marks the table as invalid.

Deletion of a table that still contains symbols throws an error. Successful deletion includes the deletion of the internal hash table and the deallocation of memory associated with the table. The table is marked invalid to prohibit any future references.

**RETURNS** OK, or ERROR if the table still contains symbols.

**SEE ALSO** VXWSymTab

---

## *VXWTask::activate()*

**NAME** *VXWTask::activate()* – activate a task (WFC Opt.)

**SYNOPSIS** `STATUS activate ()`

**DESCRIPTION** This routine activates tasks created by the form of the constructor that does not automatically activate a task. Without activation, a task is ineligible for CPU allocation by the scheduler.

**RETURNS** OK, or ERROR if the task cannot be activated.

**SEE ALSO** *VXWTask::VXWTask()*

---

## *VXWTask::deleteForce()*

**NAME** *VXWTask::deleteForce()* – delete a task without restriction (WFC Opt.)

**SYNOPSIS** `STATUS deleteForce ()`

**DESCRIPTION** This routine deletes a task even if the task is protected from deletion. It is similar to *VXWTask::~VXWTask()*. Upon deletion, all routines specified by *taskDeleteHookAdd()* are called in the context of the deleting task.

**CAVEATS** This routine is intended as a debugging aid, and is generally inappropriate for applications. Disregarding a task's deletion protection could leave the the system in an unstable state or lead to system deadlock.

The system does not protect against simultaneous *VXWTask::deleteForce()* calls. Such a situation could leave the system in an unstable state.

**RETURNS** OK, or ERROR if the task cannot be deleted.

**SEE ALSO** *taskDeleteHookAdd()*, *VXWTask::~VXWTask()*

---

## ***VXWTask::envCreate()***

**NAME** *VXWTask::envCreate()* – create a private environment (WFC Opt.)

**SYNOPSIS**

```
STATUS envCreate
(
 int envSource
)
```

**DESCRIPTION** This routine creates a private set of environment variables for a specified task, if the environment variable task create hook is not installed.

**RETURNS** OK, or ERROR if memory is insufficient.

**SEE ALSO** *VXWTask*, *envLib*

---

## ***VXWTask::errNo()***

**NAME** *VXWTask::errNo()* – retrieve error status value (WFC Opt.)

**SYNOPSIS**

```
int errNo ()
```

**DESCRIPTION** This routine gets the error status for the task.

**RETURNS** The error status value contained in **errno**.

**SEE ALSO** *VXWTask*

---

## ***VXWTask::errNo()***

**NAME** *VXWTask::errNo()* – set error status value (WFC Opt.)

**SYNOPSIS**

```
STATUS errNo
(
 int errorValue
)
```



**DESCRIPTION** This routine sets the error status value for its task.

**RETURNS** OK.

**SEE ALSO** VXWTask

## ***VXWTask::id()***

**NAME** *VXWTask::id()* – reveal task ID (WFC Opt.)

**SYNOPSIS** `int id ()`

**DESCRIPTION** This routine reveals the task ID for its task. The task ID is necessary to call C routines that affect or inquire on a task.

**RETURNS** task ID

**SEE ALSO** VXWTask, taskLib

## ***VXWTask::info()***

**NAME** *VXWTask::info()* – get information about a task (WFC Opt.)

**SYNOPSIS** `STATUS info  
(
TASK_DESC * pTaskDesc
) const`

**DESCRIPTION** This routine fills in a specified task descriptor (**TASK\_DESC**) for its task. The information in the task descriptor is, for the most part, a copy of information kept in the task control block (**WIND\_TCB**). The **TASK\_DESC** structure is useful for common information and avoids dealing directly with the unwieldy **WIND\_TCB**.

**NOTE** Examination of **WIND\_TCBs** should be restricted to debugging aids.

**RETURNS** OK

**SEE ALSO** VXWTask

***VXWTask::isReady()***

---

***VXWTask::isReady()***

**NAME** *VXWTask::isReady()* – check if task is ready to run (WFC Opt.)

**SYNOPSIS** `BOOL isReady ()`

**DESCRIPTION** This routine tests the status field of its task to determine whether the task is ready to run.

**RETURNS** TRUE if the task is ready, otherwise FALSE.

**SEE ALSO** *VXWTask*

---

***VXWTask::isSuspended()***

**NAME** *VXWTask::isSuspended()* – check if task is suspended (WFC Opt.)

**SYNOPSIS** `BOOL isSuspended ()`

**DESCRIPTION** This routine tests the status field of its task to determine whether the task is suspended.

**RETURNS** TRUE if the task is suspended, otherwise FALSE.

**SEE ALSO** *VXWTask*

---

***VXWTask::kill()***

**NAME** *VXWTask::kill()* – send a signal to task (WFC Opt.)

**SYNOPSIS** `int kill  
(  
  int signo  
)`

**DESCRIPTION** This routine sends a signal *signo* to its task.

**RETURNS** OK (0), or ERROR (-1) if the signal number is invalid.

**ERRNO**           EINVAL  
**SEE ALSO**        VXWTask

## ***VXWTask::name()***

**NAME**            ***VXWTask::name()*** – get the name associated with a task ID (WFC Opt.)

**SYNOPSIS**        **char \* name ()**

**DESCRIPTION**    This routine returns a pointer to the name of its task, if it has a name; otherwise it returns NULL.

**RETURNS**        A pointer to the task name, or NULL.

**SEE ALSO**        VXWTask

## ***VXWTask::options()***

**NAME**            ***VXWTask::options()*** – examine task options (WFC Opt.)

**SYNOPSIS**        **STATUS options**  
                  (  
                  **int \* pOptions**  
                  ) **const**

**DESCRIPTION**    This routine gets the current execution options of its task. The option bits returned indicate the following modes:

**VX\_FP\_TASK**  
                  execute with floating-point coprocessor support.

**VX\_PRIVATE\_ENV**  
                  include private environment support (see **envLib**).

**VX\_NO\_STACK\_FILL**  
                  do not fill the stack for use by **checkstack()**.

**VX\_UNBREAKABLE**  
                  do not allow breakpoint debugging.

For definitions, see **taskLib.h**.

***VXWTask::options()***

**RETURNS** OK.

**SEE ALSO** VXWTask

---

***VXWTask::options()***

**NAME** *VXWTask::options()* – change task options (WFC Opt.)

**SYNOPSIS**

```
STATUS options
(
 int mask,
 int newOptions
)
```

**DESCRIPTION** This routine changes the execution options of its task. The only option that can be changed after a task has been created is:

VX\_UNBREAKABLE – do not allow breakpoint debugging.

For definitions, see **taskLib.h**.

**RETURNS** OK.

**SEE ALSO** VXWTask

---

***VXWTask::priority()***

**NAME** *VXWTask::priority()* – examine the priority of task (WFC Opt.)

**SYNOPSIS**

```
STATUS priority
(
 int * pPriority
) const
```

**DESCRIPTION** This routine reports the current priority of its task. The current priority is copied to the integer pointed to by *pPriority*.

**RETURNS** OK.

**SEE ALSO** VXWTask

---

## ***VXWTask::priority()***

**NAME** *VXWTask::priority()* – change the priority of a task (WFC Opt.)

**SYNOPSIS**

```
STATUS priority
(
 int newPriority
)
```

**DESCRIPTION** This routine changes its task's priority to a specified priority. Priorities range from 0, the highest priority, to 255, the lowest priority.

**RETURNS** OK.

**SEE ALSO** *VXWTask*

---

## ***VXWTask::registers()***

**NAME** *VXWTask::registers()* – set a task's registers (WFC Opt.)

**SYNOPSIS**

```
STATUS registers
(
 const REG_SET * pRegs
)
```

**DESCRIPTION** This routine loads a specified register set *pRegs* into the task's TCB.

**NOTE** This routine only works well if the task is known not to be in the ready state. Suspending the task before changing the register set is recommended.

**RETURNS** OK.

**SEE ALSO** *VXWTask::suspend()*

***VXWTask::registers()***

---

## ***VXWTask::registers()***

|                    |                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>VXWTask::registers()</i> – get task registers from the TCB (WFC Opt.)                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>STATUS registers (     REG_SET * pRegs ) const</pre>                                                                                                                 |
| <b>DESCRIPTION</b> | This routine gathers task information kept in the TCB. It copies the contents of the task's registers to the register structure <i>pRegs</i> .                            |
| <b>NOTE</b>        | This routine only works well if the task is known to be in a stable, non-executing state. Self-examination, for instance, is not advisable, as results are unpredictable. |
| <b>RETURNS</b>     | OK.                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <i>VXWTask::suspend()</i>                                                                                                                                                 |

---

## ***VXWTask::restart()***

|                    |                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>VXWTask::restart()</i> – restart task (WFC Opt.)                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>STATUS restart ()</pre>                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b> | This routine "restarts" its task. The task is first terminated, and then reinitialized with the same ID, priority, options, original entry point, stack size, and parameters it had when it was terminated. Self-restarting of a calling task is performed by the exception task. |
| <b>NOTE</b>        | If the task has modified any of its start-up parameters, the restarted task will start with the changed values.                                                                                                                                                                   |
| <b>RETURNS</b>     | OK, or ERROR if the task could not be restarted.                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <i>VXWTask</i>                                                                                                                                                                                                                                                                    |

---

## **VXWTask::resume()**

|                    |                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWTask::resume()</b> – resume task (WFC Opt.)                                                   |
| <b>SYNOPSIS</b>    | <code>STATUS resume ()</code>                                                                       |
| <b>DESCRIPTION</b> | This routine resumes its task. Suspension is cleared, and the task operates in the remaining state. |
| <b>RETURNS</b>     | OK, or ERROR if the task cannot be resumed.                                                         |
| <b>SEE ALSO</b>    | VXWTask                                                                                             |

---

## **VXWTask::show()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |      |        |      |        |      |          |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|--------|------|--------|------|----------|------|---|------|--------|------|--------|------|---|------|----------|------|---|------|---|------|-------|------|-------|------|--------|------|---|------|--------|------|--------|------|------|------|-------|--|--|--|--|
| <b>NAME</b>        | <b>VXWTask::show()</b> – display the contents of task registers (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |      |        |      |        |      |          |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| <b>SYNOPSIS</b>    | <code>void show ()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |      |        |      |        |      |          |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| <b>DESCRIPTION</b> | This routine displays the register contents of its task on standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |      |        |      |        |      |          |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| <b>EXAMPLE</b>     | The following shell command line displays the register of a task <b>vxwT28</b> :<br><br><code>-&gt; vxwT28.show ()</code><br><br>The example prints on standard output a display like the following (68000 family):<br><br><table><tr><td>d0 =</td><td>0</td><td>d1 =</td><td>0</td><td>d2 =</td><td>578fe</td><td>d3 =</td><td>1</td></tr><tr><td>d4 =</td><td>3e84e1</td><td>d5 =</td><td>3e8568</td><td>d6 =</td><td>0</td><td>d7 =</td><td>ffffffff</td></tr><tr><td>a0 =</td><td>0</td><td>a1 =</td><td>0</td><td>a2 =</td><td>4f06c</td><td>a3 =</td><td>578d0</td></tr><tr><td>a4 =</td><td>3fffc4</td><td>a5 =</td><td>0</td><td>fp =</td><td>3e844c</td><td>sp =</td><td>3e842c</td></tr><tr><td>sr =</td><td>3000</td><td>pc =</td><td>4f0f2</td><td></td><td></td><td></td><td></td></tr></table> | d0 = | 0      | d1 = | 0      | d2 = | 578fe    | d3 = | 1 | d4 = | 3e84e1 | d5 = | 3e8568 | d6 = | 0 | d7 = | ffffffff | a0 = | 0 | a1 = | 0 | a2 = | 4f06c | a3 = | 578d0 | a4 = | 3fffc4 | a5 = | 0 | fp = | 3e844c | sp = | 3e842c | sr = | 3000 | pc = | 4f0f2 |  |  |  |  |
| d0 =               | 0                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | d1 = | 0      | d2 = | 578fe  | d3 = | 1        |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| d4 =               | 3e84e1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | d5 = | 3e8568 | d6 = | 0      | d7 = | ffffffff |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| a0 =               | 0                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | a1 = | 0      | a2 = | 4f06c  | a3 = | 578d0    |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| a4 =               | 3fffc4                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | a5 = | 0      | fp = | 3e844c | sp = | 3e842c   |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| sr =               | 3000                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | pc = | 4f0f2  |      |        |      |          |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |      |        |      |        |      |          |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |
| <b>SEE ALSO</b>    | VXWTask                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |      |        |      |        |      |          |      |   |      |        |      |        |      |   |      |          |      |   |      |   |      |       |      |       |      |        |      |   |      |        |      |        |      |      |      |       |  |  |  |  |

**VXWTask::show()**

---

**VXWTask::show()****NAME** `VXWTask::show()` – display task information from TCBs (WFC Opt.)**SYNOPSIS**  

```
STATUS show
(
 int level
) const
```

**DESCRIPTION** This routine displays the contents of its task's task control block (TCB). If *level* is 1, it also displays task options and registers. If *level* is 2, it displays all tasks.

The TCB display contains the following fields:

| Field  | Meaning                                                                    |
|--------|----------------------------------------------------------------------------|
| NAME   | Task name                                                                  |
| ENTRY  | Symbol name or address where task began execution                          |
| TID    | Task ID                                                                    |
| PRI    | Priority                                                                   |
| STATUS | Task status, as formatted by <code>taskStatusString()</code>               |
| PC     | Program counter                                                            |
| SP     | Stack pointer                                                              |
| ERRNO  | Most recent error code for this task                                       |
| DELAY  | If task is delayed, number of clock ticks remaining in delay (0 otherwise) |

**EXAMPLE** The following example shows the TCB contents for a task named **t28**:

```

NAME ENTRY TID PRI STATUS PC SP ERRNO DELAY

t28 _appStart 20efcac 1 READY 201dc90 20ef980 0 0
stack: base 0x20efcac end 0x20ed59c size 9532 high 1452 margin 8080
options: 0x1e
VX_UNBREAKABLE VX_DEALLOC_STACK VX_FP_TASK VX_STDIO
D0 = 0 D4 = 0 A0 = 0 A4 = 0
D1 = 0 D5 = 0 A1 = 0 A5 = 203a084 SR = 3000
D2 = 0 D6 = 0 A2 = 0 A6 = 20ef9a0 PC = 2038614
D3 = 0 D7 = 0 A3 = 0 A7 = 20ef980
```

**RETURNS** N/A**SEE ALSO** `VXWTaskstatusString()`, *Tornado User's Guide: The Tornado Shell*



## VXWTask::sigqueue()

**NAME** VXWTask::sigqueue() – send a queued signal to task (WFC Opt.)

**SYNOPSIS**

```
int sigqueue
(
 int signo,
 const union sigval value
)
```

**DESCRIPTION** The routine *sigqueue()* sends to its task the signal specified by *signo* with the signal-parameter value specified by *value*.

**RETURNS** OK (0), or ERROR (-1) if the signal number is invalid, or if there are no queued-signal buffers available.

**ERRNO** EINVAL EAGAIN

**SEE ALSO** VXWTask

## VXWTask::SRSet()

**NAME** VXWTask::SRSet() – set the task status register (MC680x0, MIPS, i386/i486) (WFC Opt.)

**SYNOPSIS**

```
STATUS SRSet
(
 UINT16 sr
)
```

**SYNOPSIS (i80X86)**

```
STATUS SRSet
(
 UINT sr
)
```

**SYNOPSIS (MIPS)**

```
STATUS SRSet
(
 UINT32 sr
)
```

**VXWTask::statusString()**

This routine sets the status register of a task that is not running; that is, you must not call this `>SRSet()`. Debugging facilities use this routine to set the trace bit in the status register of a task that is being single-stepped.

**RETURNS** OK.

**SEE ALSO** VXWTask

---

## VXWTask::statusString()

**NAME** VXWTask::statusString() – get task status as a string (WFC Opt.)

**SYNOPSIS**

```
STATUS statusString
(
 char * pString
) const
```

**DESCRIPTION** This routine deciphers the WIND task status word in the TCB for its task, and copies the appropriate string to *pString*.

The formatted string is one of the following:

| String   | Meaning                                                                    |
|----------|----------------------------------------------------------------------------|
| READY    | Task is not waiting for any resource other than the CPU.                   |
| PEND     | Task is blocked due to the unavailability of some resource.                |
| DELAY    | Task is asleep for some duration.                                          |
| SUSPEND  | Task is unavailable for execution (but not suspended, delayed, or pended). |
| DELAY+S  | Task is both delayed and suspended.                                        |
| PEND+S   | Task is both pended and suspended.                                         |
| PEND+T   | Task is pended with a timeout.                                             |
| PEND+S+T | Task is pended with a timeout, and also suspended.                         |
| ...+I    | Task has inherited priority (+I may be appended to any string above).      |
| DEAD     | Task no longer exists.                                                     |

**RETURNS** OK.

**SEE ALSO** VXWTask

---

## *VXWTask::suspend()*

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>VXWTask::suspend()</i> – suspend task (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <code>STATUS suspend ()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This routine suspends its task. Suspension is additive: thus, tasks can be delayed and suspended, or pended and suspended. Suspended, delayed tasks whose delays expire remain suspended. Likewise, suspended, pended tasks that unblock remain suspended only.</p> <p>Care should be taken with asynchronous use of this facility. The task is suspended regardless of its current state. The task could, for instance, have mutual exclusion to some system resource, such as the network or system memory partition. If suspended during such a time, the facilities engaged are unavailable, and the situation often ends in deadlock.</p> <p>This routine is the basis of the debugging and exception handling packages. However, as a synchronization mechanism, this facility should be rejected in favor of the more general semaphore facility.</p> |
| <b>RETURNS</b>     | OK, or ERROR if the task cannot be suspended.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <i>VXWTask</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

---

## *VXWTask::tcb()*

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <i>VXWTask::tcb()</i> – get the task control block (WFC Opt.)                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <code>WIND_TCB * tcb ()</code>                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine returns a pointer to the task control block (<code>WIND_TCB</code>) for its task. Although all task state information is contained in the TCB, users must not modify it directly. To change registers, for instance, use <i>VXWTask::registers()</i>.</p> |
| <b>RETURNS</b>     | A pointer to a <code>WIND_TCB</code> .                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <i>VXWTask</i>                                                                                                                                                                                                                                                            |

---

## VXWTask::varAdd()

**NAME** VXWTask::varAdd() – add a task variable to task (WFC Opt.)

**SYNOPSIS**

```
STATUS varAdd
(
 int * pVar
)
```

**DESCRIPTION** This routine adds a specified variable *pVar* (4-byte memory location) to its task's context. After calling this routine, the variable is private to the task. The task can access and modify the variable, but the modifications are not visible to other tasks, and other tasks' modifications to that variable do not affect the value seen by the task. This is accomplished by saving and restoring the variable's initial value each time a task switch occurs to or from the calling task.

This facility can be used when a routine is to be spawned repeatedly as several independent tasks. Although each task has its own stack, and thus separate stack variables, they all share the same static and global variables. To make a variable *not* shareable, the routine can call **VXWTask::varAdd()** to make a separate copy of the variable for each task, but all at the same physical address.

Note that task variables increase the task switch time to and from the tasks that own them. Therefore, it is desirable to limit the number of task variables that a task uses. One efficient way to use task variables is to have a single task variable that is a pointer to a dynamically allocated structure containing the task's private data.

**EXAMPLE** Assume that three identical tasks are spawned with a main routine called *operator()*. All three use the structure **OP\_GLOBAL** for all variables that are specific to a particular incarnation of the task. The following code fragment shows how this is set up:

```
OP_GLOBAL *opGlobal; // ptr to operator task's global variables
VXWTask me; // task object for self
void operator
(
 int opNum // number of this operator task
)
{
 me = VXWTask (0); // task object for running task
 if (me.varAdd ((int *)&opGlobal) != OK)
 {
 printErr ("operator%d: can't VXWTask::varAdd opGlobal\n", opNum);
 me.suspend ();
 }
 if ((opGlobal = (OP_GLOBAL *) malloc (sizeof (OP_GLOBAL))) == NULL)
```

```

 {
 printErr ("operator%d: can't malloc opGlobal\n", opNum);
 me.suspend ();
 }
 ...
}

```

**RETURNS** OK, or ERROR if memory is insufficient for the task variable descriptor.

**SEE ALSO** *VXWTask::varDelete()*, *VXWTask::varGet()*, *VXWTask::varSet()*

## *VXWTask::varDelete()*

**NAME** *VXWTask::varDelete()* – remove a task variable from task (WFC Opt.)

**SYNOPSIS**

```

STATUS varDelete
(
 int * pVar
)

```

**DESCRIPTION** This routine removes a specified task variable, *pVar*, from its task’s context. The private value of that variable is lost.

**RETURNS** OK, or ERROR if the task variable does not exist for the task.

**SEE ALSO** *VXWTask::varAdd()*, *VXWTask::varGet()*, *VXWTask::varSet()*

## *VXWTask::varGet()*

**NAME** *VXWTask::varGet()* – get the value of a task variable (WFC Opt.)

**SYNOPSIS**

```

int varGet
(
 int * pVar
) const

```

**DESCRIPTION** This routine returns the private value of a task variable for its task. The task is usually not the calling task, which can get its private value by directly accessing the variable. This routine is provided primarily for debugging purposes.

***VXWTask::varInfo()***

- RETURNS** The private value of the task variable, or ERROR if the task does not own the task variable.
- SEE ALSO** *VXWTask::varAdd()*, *VXWTask::varDelete()*, *VXWTask::varSet()*

---

***VXWTask::varInfo()***

- NAME** *VXWTask::varInfo()* – get a list of task variables (WFC Opt.)
- SYNOPSIS**
- ```
int varInfo
(
    TASK_VAR varList[],
    int      maxVars
) const
```
- DESCRIPTION** This routine provides the calling task with a list of all of the task variables of its task. The unsorted array of task variables is copied to *varList*.
- CAVEATS** Kernel rescheduling is disabled while task variables are looked up. There is no guarantee that all the task variables are still valid or that new task variables have not been created by the time this routine returns.
- RETURNS** The number of task variables in the list.
- SEE ALSO** *VXWTask*

VXWTask::varSet()

- NAME** *VXWTask::varSet()* – set the value of a task variable (WFC Opt.)
- SYNOPSIS**
- ```
STATUS varSet
(
 int * pVar,
 int value
)
```

**DESCRIPTION** This routine sets the private value of the task variable for a specified task. The specified task is usually not the calling task, which can set its private value by directly modifying the variable. This routine is provided primarily for debugging purposes.

**RETURNS** OK, or ERROR if the task does not own the task variable.

**SEE ALSO** *VXWTask::varAdd()*, *VXWTask::varDelete()*, *VXWTask::varGet()*

## **VXWTask::VXWTask()**

**NAME** *VXWTask::VXWTask()* – initialize a task object (WFC Opt.)

**SYNOPSIS**

```
VXWTask
(
 int tid
)
```

**DESCRIPTION** This constructor creates a task object from the task ID of an existing task. Because of the VxWorks convention that a task ID of 0 refers to the calling task, this constructor can be used to derive a task object for the calling task, as follows:

```
myTask = VXWTask (0);
```

**RETURNS** N/A

**SEE ALSO** *taskLib*, *VXWTask::~~VXWTask()*, *sp()*

## **VXWTask::VXWTask()**

**NAME** *VXWTask::VXWTask()* – create and spawn a task (WFC Opt.)

**SYNOPSIS**

```
VXWTask
(
 char * name,
 int priority,
 int options,
 int stackSize,
 FUNCPTR entryPoint,
 int arg1=0,
```

**VXWTask::VXWTask()**

```

 int arg2=0,
 int arg3=0,
 int arg4=0,
 int arg5=0,
 int arg6=0,
 int arg7=0,
 int arg8=0,
 int arg9=0,
 int arg10=0
)

```

**DESCRIPTION**

This constructor creates and activates a new task with a specified priority and options.

A task may be assigned a name as a debugging aid. This name appears in displays generated by various system information facilities such as *i()*. The name may be of arbitrary length and content, but the current VxWorks convention is to limit task names to ten characters and prefix them with a "t". If *name* is specified as NULL, an ASCII name is assigned to the task of the form "tn" where *n* is an integer which increments as new tasks are spawned.

The only resource allocated to a spawned task is a stack of a specified size *stackSize*, which is allocated from the system memory partition. Stack size should be an even integer. A task control block (TCB) is carved from the stack, as well as any memory required by the task name. The remaining memory is the task's stack and every byte is filled with the value 0xEE for the *checkStack()* facility. See the manual entry for *checkStack()* for stack-size checking aids.

The entry address *entryPt* is the address of the "main" routine of the task. The routine is called after the C environment is set up. The specified routine is called with the ten arguments provided. Should the specified main routine return, a call to *exit()* is made automatically.

Note that ten (and only ten) arguments must be passed for the spawned function.

Bits in the options argument may be set to run with the following modes:

**VX\_FP\_TASK**

execute with floating-point coprocessor support.

**VX\_PRIVATE\_ENV**

include private environment support.

**VX\_NO\_STACK\_FILL**

do not fill the stack for use by *checkstack()*.

**VX\_UNBREAKABLE**

do not allow breakpoint debugging.

See the definitions in **taskLib.h**.

**RETURNS**

N/A



**SEE ALSO** *VXWTask::~VXWTask()*, *VXWTask::activate()*, *sp()*, *VxWorks Programmer's Guide: Basic OS*

---

## VXWTask::VXWTask()

**NAME** *VXWTask::VXWTask()* – initialize a task with a specified stack (WFC Opt.)

**SYNOPSIS**

```
VXWTask
(
 WIND_TCB * pTcb,
 char * name,
 int priority,
 int options,
 char * pStackBase,
 int stackSize,
 FUNCPTR entryPoint,
 int arg1=0,
 int arg2=0,
 int arg3=0,
 int arg4=0,
 int arg5=0,
 int arg6=0,
 int arg7=0,
 int arg8=0,
 int arg9=0,
 int arg10=0
)
```

**DESCRIPTION** This constructor initializes user-specified regions of memory for a task stack and control block instead of allocating them from memory. This constructor uses the specified pointers to the `WIND_TCB` and stack as the components of the task. This allows, for example, the initialization of a static `WIND_TCB` variable. It also allows for special stack positioning as a debugging aid.

As in other constructors, a task may be given a name. If no name is specified, this constructor creates a task without a name (rather than assigning a default name).

Other arguments are the same as in the previous constructor. This constructor does not activate the task. This must be done by calling *VXWTask::activate()*.

Normally, tasks should be started using the previous constructor rather than this one, except when additional control is required for task memory allocation or a separate task activation is desired.

## ***VXWTask::~~VXWTask()***

**RETURNS** OK, or ERROR if the task cannot be initialized.

**SEE ALSO** *VXWTask::activate()*

---

## ***VXWTask::~~VXWTask()***

**NAME** *VXWTask::~~VXWTask()* – delete a task (WFC Opt.)

**SYNOPSIS** `virtual ~VXWTask ()`

**DESCRIPTION** This destructor causes the task to cease to exist and deallocates the stack and **WIND\_TCB** memory resources. Upon deletion, all routines specified by *taskDeleteHookAdd()* are called in the context of the deleting task.

**RETURNS** N/A

**SEE ALSO** *excLib*, *taskDeleteHookAdd()*, *VXWTask::VXWTask()*, *VxWorks Programmer's Guide: Basic OS*

---

## ***VXWWd::cancel()***

**NAME** *VXWWd::cancel()* – cancel a currently counting watchdog (WFC Opt.)

**SYNOPSIS** `STATUS cancel ()`

**DESCRIPTION** This routine cancels a currently running watchdog timer by zeroing its delay count. Watchdog timers may be canceled from interrupt level.

**RETURNS** OK, or ERROR if the watchdog timer cannot be canceled.

**SEE ALSO** *VXWWd::start()*

---

## VXWwd::start()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWwd::start()</b> – start a watchdog timer (WFC Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>STATUS start (     int    delay,     FUNCPTR pRoutine,     int    parameter )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>This routine adds a watchdog timer to the system tick queue. The specified watchdog routine will be called from interrupt level after the specified number of ticks has elapsed. Watchdog timers may be started from interrupt level.</p> <p>To replace either the timeout <i>delay</i> or the routine to be executed, call <b>VXWwd::start()</b> again; only the most recent <b>VXWwd::start()</b> on a given watchdog ID has any effect. (If your application requires multiple watchdog routines, use <b>VXWwd::VXWwd()</b> to generate separate a watchdog for each.) To cancel a watchdog timer before the specified tick count is reached, call <b>VXWwd::cancel()</b>.</p> <p>Watchdog timers execute only once, but some applications require periodically executing timers. To achieve this effect, the timer routine itself must call <b>VXWwd::start()</b> to restart the timer on each invocation.</p> |
| <b>WARNING</b>     | The watchdog routine runs in the context of the system-clock ISR; thus, it is subject to all ISR restrictions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>RETURNS</b>     | OK, or ERROR if the watchdog timer cannot be started.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>VXWwd::cancel()</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

---

## VXWwd::VXWwd()

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <b>NAME</b>        | <b>VXWwd::VXWwd()</b> – construct a watchdog timer (WFC Opt.) |
| <b>SYNOPSIS</b>    | <pre>vxwwd ()</pre>                                           |
| <b>DESCRIPTION</b> | This routine creates a watchdog timer.                        |
| <b>RETURNS</b>     | N/A                                                           |
| <b>SEE ALSO</b>    | <b>VXWwd::~VXWwd()</b>                                        |

**VXWwd::VXWwd()**

---

**VXWwd::VXWwd()**

**NAME** VXWwd::VXWwd() – construct a watchdog timer (WFC Opt.)

**SYNOPSIS**

```
VXWwd
(
 WDOG_ID aWdId
)
```

**DESCRIPTION** This routine creates a watchdog timer from an existing WDOG\_ID.

**RETURNS** N/A

**SEE ALSO** VXWwd::~~VXWwd()

---

**VXWwd::~~VXWwd()**

**NAME** VXWwd::~~VXWwd() – destroy a watchdog timer (WFC Opt.)

**SYNOPSIS**

```
~VXWwd ()
```

**DESCRIPTION** This routine destroys a watchdog timer. The watchdog will be removed from the timer queue if it has been started.

**RETURNS** N/A

**SEE ALSO** VXWwd::VXWwd()

---

## *wcstombs()*

|                      |                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>wcstombs()</i> – convert a series of wide char’s to multibyte char’s (Unimplemented) (ANSI)        |
| <b>SYNOPSIS</b>      | <pre>size_t wcstombs (     char *          s,     const wchar_t * pwcs,     size_t          n )</pre> |
| <b>DESCRIPTION</b>   | This multibyte character function is unimplemented in VxWorks.                                        |
| <b>INCLUDE FILES</b> | <code>stdlib.h</code>                                                                                 |
| <b>RETURNS</b>       | OK, or ERROR if the parameters are invalid.                                                           |
| <b>SEE ALSO</b>      | <code>ansiStdlib</code>                                                                               |

---

## *wctomb()*

|                      |                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <i>wctomb()</i> – convert a wide character to a multibyte character (Unimplemented) (ANSI) |
| <b>SYNOPSIS</b>      | <pre>int wctomb (     char * s,     wchar_t wchar )</pre>                                  |
| <b>DESCRIPTION</b>   | This multibyte character function is unimplemented in VxWorks.                             |
| <b>INCLUDE FILES</b> | <code>stdlib.h</code>                                                                      |
| <b>RETURNS</b>       | OK, or ERROR if the parameters are invalid.                                                |
| <b>SEE ALSO</b>      | <code>ansiStdlib</code>                                                                    |

---

## *wd33c93CtrlCreate()*

**NAME** *wd33c93CtrlCreate()* – create and partially initialize a WD33C93 SBIC structure

**SYNOPSIS**

```
WD_33C93_SCSI_CTRL *wd33c93CtrlCreate
(
 UINT8 * sbicBaseAdrs, /* base address of SBIC */
 int regOffset, /* addr offset between consecutive regs. */
 UINT clkPeriod, /* period of controller clock (nsec) */
 int devType, /* SBIC device type */
 FUNCPTR sbicScsiReset, /* SCSI bus reset function */
 FUNCPTR sbicDmaBytesIn, /* SCSI DMA input function */
 FUNCPTR sbicDmaBytesOut /* SCSI DMA output function */
)
```

**DESCRIPTION** This routine creates an SBIC data structure and must be called before using an SBIC chip. It should be called once and only once for a specified SBIC. Since it allocates memory for a structure needed by all routines in **wd33c93Lib**, it must be called before any other routines in the library. After calling this routine, at least one call to *wd33c93CtrlInit()* should be made before any SCSI transaction is initiated using the SBIC.

Note that only the non-multiplexed processor interface is supported.

The input parameters are as follows:

*sbicBaseAdrs*

the address where the CPU accesses the lowest register of the SBIC.

*regOffset*

the address offset (in bytes) to access consecutive registers. (This must be a power of 2; for example, 1, 2, 4, etc.)

*clkPeriod*

the period, in nanoseconds, of the signal-to-SBIC clock input used only for select command timeouts.

*devType*

a constant corresponding to the type (part number) of this controller; possible options are enumerated in **wd33c93.h** under the heading "SBIC device type."

*sbicScsiReset*

a board-specific routine to assert the RST line on the SCSI bus, which causes all connected devices to return to a known quiescent state.

*spcDmaBytesIn* and *spcDmaBytesOut*

board-specific routines to handle DMA input and output. If these are NULL (0), SBIC program transfer mode is used. DMA is implemented only during SCSI data in/out phases. The interface to these DMA routines must be of the form:

```

STATUS xxDmaBytes{In, Out}
(
 SCSI_PHYS_DEV *pScsiPhysDev, /* ptr to phys dev info */
 UINT8 *pBuffer, /* ptr to the data buffer */
 int bufLength /* number of bytes to xfer */
)

```

**RETURNS** A pointer to the SBIC control structure, or NULL if memory is insufficient or parameters are invalid.

**SEE ALSO** wd33c93Lib1, wd33c93.h

## *wd33c93CtrlCreateScsi2()*

**NAME** *wd33c93CtrlCreateScsi2()* – create and partially initialize an SBIC structure

**SYNOPSIS**

```

WD_33C93_SCSI_CTRL *wd33c93CtrlCreateScsi2
(
 UINT8 * sbicBaseAdrs, /* base address of the SBIC */
 int regOffset, /* address offset between SBIC registers */
 UINT clkPeriod, /* period of the SBIC clock (nsec) */
 FUNCPTR sysScsiBusReset, /* function to reset SCSI bus */
 int sysScsiResetArg, /* argument to pass to above function */
 UINT sysScsiDmaMaxBytes, /* maximum byte count using DMA */
 FUNCPTR sysScsiDmaStart, /* function to start SCSI DMA transfer */
 FUNCPTR sysScsiDmaAbort, /* function to abort SCSI DMA transfer */
 int sysScsiDmaArg /* argument to pass to above functions */
)

```

**DESCRIPTION** This routine creates an SBIC data structure and must be called before using an SBIC chip. It must be called exactly once for a specified SBIC. Since it allocates memory for a structure needed by all routines in **wd33c93Lib2**, it must be called before any other routines in the library. After calling this routine, at least one call to *wd33c93CtrlInit()* must be made before any SCSI transaction is initiated using the SBIC.

**NOTE** Only the non-multiplexed processor interface is supported.

A detailed description of the input parameters follows:

**sbicBaseAdrs**

the address at which the CPU would access the lowest (AUX STATUS) register of the SBIC.

**regOffset**

the address offset (bytes) to access consecutive registers. (This must be a power of 2, for example, 1, 2, 4, etc.)

**clkPeriod**

the period in nanoseconds of the signal to SBIC CLK input.

**sysScsiBusReset** and **sysScsiResetArg**

the board-specific routine to pulse the SCSI bus RST signal. The specified argument is passed to this routine when it is called. It may be used to identify the SCSI bus to be reset, if there is a choice. The interface to this routine is of the form:

```
void xxBusReset
(
 int arg; /* call-back argument */
)
```

**sysScsiDmaMaxBytes**, **sysScsiDmaStart**, **sysScsiDmaAbort**, and **sysScsiDmaArg**

board-specific routines to handle DMA transfers to and from the SBIC; if the maximum DMA byte count is zero, programmed I/O is used. Otherwise, non-NULL function pointers to DMA start and abort routines must be provided. The specified argument is passed to these routines when they are called; it may be used to identify the DMA channel to use, for example. Note that DMA is implemented only during SCSI data in/out phases. The interface to these DMA routines must be of the form:

```
STATUS xxDmaStart
(
 int arg; /* call-back argument */
 UINT8 *pBuffer; /* ptr to the data buffer */
 UINT buflength; /* number of bytes to xfer */
 int direction; /* 0 = SCSI->mem, 1 = mem->SCSI */
)
STATUS xxDmaAbort
(
 int arg; /* call-back argument */
)
```

**RETURNS**

A pointer to the SBIC structure, or NULL if memory is insufficient or the parameters are invalid.

**SEE ALSO**

**wd33c93Lib2**



---

## *wd33c93CtrlInit()*

**NAME** *wd33c93CtrlInit()* – initialize the user-specified fields in an SBIC structure

**SYNOPSIS**

```
STATUS wd33c93CtrlInit
(
 int * pSbic, /* ptr to SBIC info */
 int scsiCtrlBusId, /* SCSI bus ID of this SBIC */
 UINT defaultSelTimeout, /* default dev. select timeout (microsec) */
 int scsiPriority /* priority of task when doing SCSI I/O */
)
```

**DESCRIPTION** This routine initializes an SBIC structure, after the structure is created with either *wd33c93CtrlCreate()* or *wd33c93CtrlCreateScsi2()*. This structure must be initialized before the SBIC can be used. It may be called more than once; however, it should be called only while there is no activity on the SCSI interface.

Before returning, this routine pulses RST (reset) on the SCSI bus, thus resetting all attached devices.

The input parameters are as follows:

*pSbic*

a pointer to the `WD_33C93_SCSI_CTRL` structure created with *wd33c93CtrlCreate()* or *wd33c93CtrlCreateScsi2()*.

*scsiCtrlBusId*

the SCSI bus ID of the SBIC, in the range 0 – 7. The ID is somewhat arbitrary; the value 7, or highest priority, is conventional.

*defaultSelTimeout*

the timeout, in microseconds, for selecting a SCSI device attached to this controller. This value is used as a default if no timeout is specified in *scsiPhysDevCreate()*. The recommended value zero (0) specifies `SCSI_DEF_SELECT_TIMEOUT` (250 millisec). The maximum timeout possible is approximately 2 seconds. Values exceeding this revert to the maximum. For more information about chip timeouts, see the manuals *Western Digital WD33C92/93 SCSI-Bus Interface Controller*, *Western Digital WD33C92A/93A SCSI-Bus Interface Controller*.

*scsiPriority*

the priority to which a task is set when performing a SCSI transaction. Valid priorities are 0 to 255. Alternatively, the value -1 specifies that the priority should not be altered during SCSI transactions.

**RETURNS** OK, or ERROR if a parameter is out of range.

**SEE ALSO** *wd33c93Lib*, *scsiPhysDevCreate()*, *Western Digital WD33C92/93 SCSI-Bus Interface Controller*, *Western Digital WD33C92A/93A SCSI-Bus Interface Controller*

---

## *wd33c93Show()*

**NAME** *wd33c93Show()* – display the values of all readable WD33C93 chip registers

**SYNOPSIS**

```
int wd33c93Show
(
 int * pScsiCtrl /* ptr to SCSI controller info */
)
```

**DESCRIPTION** This routine displays the state of the SBIC registers in a user-friendly manner. It is useful primarily for debugging. It should not be invoked while another running process is accessing the SCSI controller.

**EXAMPLE**

```
-> wd33c93Show
REG #00 (Own ID) = 0x07
REG #01 (Control) = 0x00
REG #02 (Timeout Period) = 0x20
REG #03 (Sectors) = 0x00
REG #04 (Heads) = 0x00
REG #05 (Cylinders MSB) = 0x00
REG #06 (Cylinders LSB) = 0x00
REG #07 (Log. Addr. MSB) = 0x00
REG #08 (Log. Addr. 2SB) = 0x00
REG #09 (Log. Addr. 3SB) = 0x00
REG #0a (Log. Addr. LSB) = 0x00
REG #0b (Sector Number) = 0x00
REG #0c (Head Number) = 0x00
REG #0d (Cyl. Number MSB) = 0x00
REG #0e (Cyl. Number LSB) = 0x00
REG #0f (Target LUN) = 0x00
REG #10 (Command Phase) = 0x00
REG #11 (Synch. Transfer) = 0x00
REG #12 (Xfer Count MSB) = 0x00
REG #13 (Xfer Count 2SB) = 0x00
REG #14 (Xfer Count LSB) = 0x00
REG #15 (Destination ID) = 0x03
REG #16 (Source ID) = 0x00
REG #17 (SCSI Status) = 0x42
REG #18 (Command) = 0x07
```

**RETURNS** OK, or ERROR if *pScsiCtrl* and *pSysScsiCtrl* are both NULL.

**SEE ALSO** wd33c93Lib

2

---

## *wdbNetromPktDevInit()*

**NAME** *wdbNetromPktDevInit()* – initialize a NETROM packet device for the WDB agent

**SYNOPSIS**

```
void wdbNetromPktDevInit
(
 WDB_NETROM_PKT_DEV * pPktDev, /* packet device to initialize */
 caddr_t dpBase, /* address of dualport memory */
 int width, /* number of bytes in a ROM word */
 int index, /* pod zero's index in a ROM word */
 int numAccess, /* to pod zero per byte read */
 void (* stackRcv)(), /* callback when packet arrives */
 int pollDelay /* poll task delay */
)
```

**DESCRIPTION** This routine initializes a NETROM packet device. It is typically called from **usrWdb.c** when the WDB agents NETROM communication path is selected. The *dpBase* parameter is the address of NetROM's dualport RAM. The *width* parameter is the width of a word in ROM space, and can be 1, 2, or 4 to select 8-bit, 16-bit, or 32-bit width respectively (use the macro **WDB\_NETROM\_WIDTH** in **configAll.h** for this parameter). The *index* parameter refers to which byte of the ROM contains pod zero. The *numAccess* parameter should be set to the number of accesses to POD zero that are required to read a byte. It is typically one, but some boards actually read a word at a time. This routine spawns a task which polls the NetROM for incoming packets every *pollDelay* clock ticks.

**RETURNS** N/A

**SEE ALSO** wdbNetromPktDrv

**wdbPipePktDevInit()**

---

**wdbPipePktDevInit()**

**NAME** *wdbPipePktDevInit()* – initialize a pipe packet device.

**SYNOPSIS** `STATUS wdbPipePktDevInit`

```
(
 WDB_PIPE_PKT_DEV * pPktDev, /* pipe device structure to init */
 void (* stackRcv)() /* receive packet callback (udpRcv) */
)
```

**SEE ALSO** `wdbPipePktDrv`

---

**wdbSlipPktDevInit()**

**NAME** *wdbSlipPktDevInit()* – initialize a SLIP packet device for a WDB agent

**SYNOPSIS** `void wdbSlipPktDevInit`

```
(
 WDB_SLIP_PKT_DEV * pPktDev, /* SLIP packetizer device */
 SIO_CHAN * pSioChan, /* underlying serial channel */
 void (* stackRcv)() /* callback when a packet arrives */
)
```

**DESCRIPTION** This routine initializes a SLIP packet device on one of the BSP's serial channels. It is typically called from `usrWdb.c` when the WDB agent's lightweight SLIP communication path is selected.

**RETURNS** N/A

**SEE ALSO** `wdbSlipPktDrv`

---

**wdbSystemSuspend()**

**NAME** *wdbSystemSuspend()* – suspend the system.

**SYNOPSIS** `STATUS wdbSystemSuspend (void)`

**DESCRIPTION** This routine transfers control from the run time system to the WDB agent running in external mode. In order to give back the control to the system it must be resumed by the the external WDB agent.

**EXAMPLE** The code below, called in a vxWorks application, suspends the system :

```
if (wdbSystemSuspend != OK)
 printf ("External mode is not supported by the WDB agent.\n");
```

From a host tool, we can detect that the system is suspended.

First, attach to the target server :

```
wtxtcl> wtxToolAttach EP960CX
EP960CX_ps@sevre
```

Then, you can get the agent mode :

```
wtxtcl> wtxAgentModeGet
AGENT_MODE_EXTERN
```

To get the status of the system context, execute :

```
wtxtcl> wtxContextStatusGet CONTEXT_SYSTEM 0
CONTEXT_SUSPENDED
```

In order to resume the system, simply execute :

```
wtxtcl> wtxContextResume CONTEXT_SYSTEM 0
0
```

You will see that the system is now running :

```
wtxtcl> wtxContextStatusGet CONTEXT_SYSTEM 0
CONTEXT_RUNNING
```

**RETURNS** OK upon successful completion, ERROR if external mode is not supported by the WDB agent.

**SEE ALSO** **wdbLib**

---

## *wdbTsfsDrv()*

- NAME** *wdbTsfsDrv()* – initialize the TSFS device driver for a WDB agent
- SYNOPSIS**
- ```
STATUS wdbTsfsDrv
(
    char * name /* root name in i/o system */
)
```
- DESCRIPTION**
- This routine initializes the VxWorks virtual I/O "2" driver and creates a TSFS device of the specified name.
- This routine should be called exactly once, before any reads, writes, or opens. Normally, it is called by *usrRoot()* in *usrConfig.c*, and the device name created is */tgtsvr*.
- After this routine has been called, individual virtual I/O channels can be opened by appending the host file name to the virtual I/O device name. For example, to get a file descriptor for the host file */etc/passwd*, call *open()* as follows:
- ```
fd = open ("/tgtsvr/etc/passwd", O_RDWR, 0)
```
- RETURNS** OK, or ERROR if the driver can not be installed.
- SEE ALSO** *wdbTsfsDrv*

---

## *wdbUlipPktDevInit()*

- NAME** *wdbUlipPktDevInit()* – initialize the WDB agent's communication functions for ULIP
- SYNOPSIS**
- ```
void wdbUlipPktDevInit
(
    WDB_ULIP_PKT_DEV * pDev,      /* ULIP packet device to initialize */
    char *            ulipDev,    /* name of UNIX device to use */
    void (*           stackRcv)() /* routine to call when a packet arrives */
)
```
- DESCRIPTION**
- This routine initializes a ULIP device for use by the WDB debug agent. It provides a communication path to the debug agent which can be used with both a task and an external mode agent. It is typically called by *usrWdb.c* when the WDB agent's lightweight ULIP communication path is selected.
- RETURNS** N/A

SEE ALSO `wdbUlipPktDrv`

wdbUserEvtLibInit()

NAME `wdbUserEvtLibInit()` – include the WDB user event library

SYNOPSIS `void wdbUserEvtLibInit (void)`

DESCRIPTION This null routine is provided so that `wdbUserEvtLib` can be linked into the system. If `INCLUDE_WDB_USER_EVENT` is defined in `configAll.h`, `wdbUserEvtLibInit` is called by the WDB config routine, `wdbConfig()`, in `usrWdb.c`.

RETURNS N/A

SEE ALSO `wdbUserEvtLib`

wdbUserEvtPost()

NAME `wdbUserEvtPost()` – post a user event string to host tools.

SYNOPSIS `STATUS wdbUserEvtPost`
`(`
`char * event /* event string to send */`
`)`

DESCRIPTION This routine posts the string *event* to host tools that have registered for it. Host tools will receive a USER WTX event string. The maximum size of the event is `WDB_MAX_USER_EVT_SIZE` (defined in `$WIND_BASE/target/h/wdb/wdbLib.h`).

EXAMPLE The code below sends a WDB user event to host tools :

```
char * message = "Alarm: reactor overheating !!!";
if (wdbUserEvtPost (message) != OK)
    printf ("Can't send alarm message to host tools");
```

This event will be received by host tools that have registered for it. For example a WTX TCL based tool would do :

```
wtxtcl> wtxToolAttach EP960CX
EP960CX_ps@sevre
```

wdbVioDrv()

```

wtxctl> wtxRegisterForEvent "USER.*"
0
wtxctl> wtxEventGet
USER Alarm: reactor overheating !!!

```

Host tools can register for more specific user events :

```

wtxctl> wtxToolAttach EP960CX
EP960CX_ps@sevre
wtxctl> wtxRegisterForEvent "USER Alarm.*"
0
wtxctl> wtxEventGet
USER Alarm: reactor overheating !!!

```

In this piece of code, only the USER events beginning with "Alarm" will be received.

- RETURNS** OK upon successful completion, a WDB error code if unable to send the event to the host or ERROR if the size of the event is greater than `WDB_MAX_USER_EVT_SIZE`.
- SEE ALSO** `wdbUserEvtLib`

wdbVioDrv()

- NAME** `wdbVioDrv()` – initialize the tty driver for a WDB agent
- SYNOPSIS**
- ```

STATUS wdbVioDrv
(
 char * name
)

```
- DESCRIPTION**
- This routine initializes the VxWorks virtual I/O driver and creates a virtual I/O device of the specified name.
- This routine should be called exactly once, before any reads, writes, or opens. Normally, it is called by `usrRoot()` in `usrConfig.c`, and the device name created is `"/vio"`.
- After this routine has been called, individual virtual I/O channels can be open by appending the channel number to the virtual I/O device name. For example, to get a file descriptor for virtual I/O channel `0x1000017`, call `open()` as follows:
- ```

fd = open ("/vio/0x1000017", O_RDWR, 0)

```
- RETURNS** OK, or ERROR if the driver cannot be installed.
- SEE ALSO** `wdbVioDrv`

wdCancel()

NAME *wdCancel()* – cancel a currently counting watchdog

SYNOPSIS

```
STATUS wdCancel
(
    WDOG_ID wdId /* ID of watchdog to cancel */
)
```

DESCRIPTION This routine cancels a currently running watchdog timer by zeroing its delay count. Watchdog timers may be canceled from interrupt level.

RETURNS OK, or ERROR if the watchdog timer cannot be canceled.

SEE ALSO *wdLib*, *wdStart()*

wdCreate()

NAME *wdCreate()* – create a watchdog timer

SYNOPSIS

```
WDOG_ID wdCreate (void)
```

DESCRIPTION This routine creates a watchdog timer by allocating a WDOG structure in memory.

RETURNS The ID for the watchdog created, or NULL if memory is insufficient.

SEE ALSO *wdLib*, *wdDelete()*

wdDelete()

NAME *wdDelete()* – delete a watchdog timer

SYNOPSIS

```
STATUS wdDelete
(
    WDOG_ID wdId /* ID of watchdog to delete */
)
```

wdShow()

- DESCRIPTION** This routine de-allocates a watchdog timer. The watchdog will be removed from the timer queue if it has been started. This routine complements *wdCreate()*.
- RETURNS** OK, or ERROR if the watchdog timer cannot be de-allocated.
- SEE ALSO** *wdLib*, *wdCreate()*

wdShow()

NAME *wdShow()* – show information about a watchdog

SYNOPSIS

```
STATUS wdShow
(
    WDOG_ID wdId /* watchdog to display */
)
```

DESCRIPTION This routine displays the state of a watchdog.

EXAMPLE A summary of the state of a watchdog is displayed as follows:

```
-> wdShow myWdId
Watchdog Id      : 0x3dd46c
State            : OUT_OF_Q
Ticks Remaining  : 0
Routine          : 0
Parameter        : 0
```

RETURNS OK or ERROR.

SEE ALSO *wdShow*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

wdShowInit()

NAME *wdShowInit()* – initialize the watchdog show facility

SYNOPSIS

```
void wdShowInit (void)
```

| | |
|--------------------|---|
| DESCRIPTION | This routine links the watchdog show facility into the VxWorks system. It is called automatically when the watchdog show facility is configured into VxWorks using either of the following methods: <ul style="list-style-type: none">– If you use the configuration header files, define <code>INCLUDE_SHOW_ROUTINES</code> in <code>config.h</code>.– If you use the Tornado project facility, select <code>INCLUDE_WATCHDOGS_SHOW</code>. |
| RETURNS | N/A |
| SEE ALSO | <code>wdShow</code> |

wdStart()

| | |
|--------------------|--|
| NAME | <i>wdStart()</i> – start a watchdog timer |
| SYNOPSIS | <pre>STATUS wdStart (WDOG_ID wdId, /* watchdog ID */ int delay, /* delay count, in ticks */ FUNCPTR pRoutine, /* routine to call on time-out */ int parameter /* parameter with which to call routine */)</pre> |
| DESCRIPTION | <p>This routine adds a watchdog timer to the system tick queue. The specified watchdog routine will be called from interrupt level after the specified number of ticks has elapsed. Watchdog timers may be started from interrupt level.</p> <p>To replace either the timeout <i>delay</i> or the routine to be executed, call <i>wdStart()</i> again with the same <i>wdId</i>; only the most recent <i>wdStart()</i> on a given watchdog ID has any effect. (If your application requires multiple watchdog routines, use <i>wdCreate()</i> to generate separate a watchdog ID for each.) To cancel a watchdog timer before the specified tick count is reached, call <i>wdCancel()</i>.</p> <p>Watchdog timers execute only once, but some applications require periodically executing timers. To achieve this effect, the timer routine itself must call <i>wdStart()</i> to restart the timer on each invocation.</p> |
| WARNING | The watchdog routine runs in the context of the system-clock ISR; thus, it is subject to all ISR restrictions. |
| RETURNS | OK, or ERROR if the watchdog timer cannot be started. |
| SEE ALSO | <code>wdLib</code> , <i>wdCancel()</i> |

whoami()

whoami()

NAME *whoami()* – display the current remote identity

SYNOPSIS `void whoami (void)`

DESCRIPTION This routine displays the user name currently used for remote machine access. The user name is set with *iam()* or *remCurIdSet()*.

RETURNS N/A

SEE ALSO *remLib*, *iam()*, *remCurIdGet()*, *remCurIdSet()*

wim()

NAME *wim()* – return the contents of the window invalid mask register (SPARC)

SYNOPSIS `int wim
(
 int taskId /* task ID, 0 means default task */
)`

DESCRIPTION This command extracts the contents of the window invalid mask register from the TCB of a specified task. If *taskId* is omitted or 0, the default task is assumed.

RETURNS The contents of the window invalid mask register.

SEE ALSO *dbgArchLib*, *VxWorks Programmer's Guide: Target Shell*

winDevInit()

NAME *winDevInit()* – initialize a WIN_CHAN

SYNOPSIS `void winDevInit
(
 WIN_CHAN * pChan
)`

| | |
|--------------------|--|
| DESCRIPTION | This routine initializes the driver function pointers and then resets the chip in a quiescent state. The BSP must have already initialized all the device addresses and the baudFreq fields in the WIN_CHAN structure before passing it to this routine. |
| RETURNS | N/A |
| SEE ALSO | <i>winSio</i> |

winDevInit2()

| | |
|--------------------|--|
| NAME | <i>winDevInit2()</i> – initialize a WIN_CHAN, part 2 |
| SYNOPSIS | <pre>void winDevInit2 (WIN_CHAN * pChan /* device to initialize */)</pre> |
| DESCRIPTION | This routine is called by the BSP after interrupts have been connected. The driver can now operate in interrupt mode. Before this routine is called only polled mode operations should be allowed. |
| RETURNS | N/A |
| SEE ALSO | <i>winSio</i> |

winIntRcv()

| | |
|--------------------|---|
| NAME | <i>winIntRcv()</i> – handle a channel's receive-character interrupt |
| SYNOPSIS | <pre>void winIntRcv (WIN_CHAN * pChan, /* channel generating the interrupt */ UINT16 wparam /* message args get passed if you look */)</pre> |
| DESCRIPTION | This function is attached to the simulator's interrupt handler, and passes the character received in the message to the callback. |
| RETURNS | N/A |

winIntTx()

SEE ALSO **winSio**

winIntTx()

NAME *winIntTx()* – transmit a single character.

SYNOPSIS

```
void winIntTx
(
    WIN_CHAN * pChan /* channel generating the interrupt */
)
```

DESCRIPTION This displays a single character to the simulator's window.

RETURNS N/A

SEE ALSO **winSio**

write()

NAME *write()* – write bytes to a file

SYNOPSIS

```
int write
(
    int    fd,          /* file descriptor on which to write */
    char * buffer,     /* buffer containing bytes to be written */
    size_t nbytes     /* number of bytes to write */
)
```

DESCRIPTION This routine writes *nbytes* bytes from *buffer* to a specified file descriptor *fd*. It calls the device driver to do the work.

RETURNS The number of bytes written (if not equal to *nbytes*, an error has occurred), or ERROR if the file descriptor does not exist, the driver does not have a write routine, or the driver returns ERROR. If the driver does not have a write routine, errno is set to ENOTSUP.

SEE ALSO **ioLib**

y()

| | |
|--------------------|---|
| NAME | <i>y()</i> – return the contents of the y register (SPARC) |
| SYNOPSIS | <pre>int y (int taskId /* task ID, 0 means default task */)</pre> |
| DESCRIPTION | This command extracts the contents of the y register from the TCB of a specified task. If <i>taskId</i> is omitted or 0, the default task is assumed. |
| RETURNS | The contents of the y register. |
| SEE ALSO | dbgArchLib , <i>VxWorks Programmer's Guide: Target Shell</i> |

z8530DevInit()

| | |
|--------------------|--|
| NAME | <i>z8530DevInit()</i> – initialize a Z8530_DUSART |
| SYNOPSIS | <pre>void z8530DevInit (Z8530_DUSART * pDusart)</pre> |
| DESCRIPTION | The BSP must have already initialized all the device addresses, etc in Z8530_DUSART structure. This routine initializes some SIO_CHAN function pointers and then resets the chip to a quiescent state. |
| RETURNS | N/A |
| SEE ALSO | z8530Sio |

z8530Int()

| | |
|--------------------|--|
| NAME | z8530Int() – handle all interrupts in one vector |
| SYNOPSIS | <pre>void z8530Int (Z8530_DUSART * pDusart)</pre> |
| DESCRIPTION | On some boards, all SCC interrupts for both ports share a single interrupt vector. This is the ISR for such boards. We determine from the parameter which SCC interrupted, then look at the code to find out which channel and what kind of interrupt. |
| RETURNS | N/A |
| SEE ALSO | z8530Sio |

z8530IntEx()

| | |
|--------------------|---|
| NAME | z8530IntEx() – handle error interrupts |
| SYNOPSIS | <pre>void z8530IntEx (Z8530_CHAN * pChan)</pre> |
| DESCRIPTION | This routine handles miscellaneous interrupts on the SCC. |
| RETURNS | N/A |
| SEE ALSO | z8530Sio |

z8530IntRd()

NAME *z8530IntRd()* – handle a receiver interrupt

SYNOPSIS

```
void z8530IntRd
(
    Z8530_CHAN * pChan
)
```

DESCRIPTION This routine handles read interrupts from the SCC.

RETURNS N/A

SEE ALSO *z8530Sio*

z8530IntWr()

NAME *z8530IntWr()* – handle a transmitter interrupt

SYNOPSIS

```
void z8530IntWr
(
    Z8530_CHAN * pChan
)
```

DESCRIPTION This routine handles write interrupts from the SCC.

RETURNS N/A

SEE ALSO *z8530Sio*

zbufCreate()

NAME *zbufCreate()* – create an empty zbuf

SYNOPSIS

```
ZBUF_ID zbufCreate (void)
```

zbufCut()

| | |
|--------------------|--|
| DESCRIPTION | This routine creates a zbuf, which remains empty (that is, it contains no data) until segments are added by the zbuf insertion routines. Operations performed on zbufs require a zbuf ID, which is returned by this routine. |
| RETURNS | A zbuf ID, or NULL if a zbuf cannot be created. |
| SEE ALSO | zbufLib , <i>zbufDelete()</i> |

zbufCut()

NAME *zbufCut()* – delete bytes from a zbuf

SYNOPSIS

```
ZBUF_SEG zbufCut
(
    ZBUF_ID  zbufId, /* zbuf from which bytes are cut */
    ZBUF_SEG zbufSeg, /* zbuf segment base for offset */
    int      offset, /* relative byte offset */
    int      len     /* number of bytes to cut */
)
```

DESCRIPTION

This routine deletes *len* bytes from *zbufId* starting at the specified byte location. The starting location of deletion is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, the first byte deleted is the exact byte specified by *zbufSeg* and *offset*.

The number of bytes to delete is given by *len*. If this parameter is negative, or is larger than the number of bytes in the zbuf after the specified byte location, the rest of the zbuf is deleted. The bytes deleted may span more than one segment.

If all the bytes in any one segment are deleted, then the segment is deleted, and the data buffer that it referenced will be freed if no other zbuf segments reference it. No segment may survive with zero bytes referenced.

Deleting bytes out of the middle of a segment splits the segment into two. The first segment contains the portion of the data buffer before the deleted bytes, while the other segment contains the end portion that remains after deleting *len* bytes.

This routine returns the zbuf segment ID of the segment just after the deleted bytes. In the case where bytes are cut off the end of a zbuf, a value of **ZBUF_NONE** is returned.

RETURNS The zbuf segment ID of the segment following the deleted bytes, or NULL if the operation fails.

SEE ALSO **zbufLib**

zbufDelete()

| | |
|--------------------|--|
| NAME | <i>zbufDelete()</i> – delete a zbuf |
| SYNOPSIS | <pre>STATUS zbufDelete (ZBUF_ID zbufId /* zbuf to be deleted */)</pre> |
| DESCRIPTION | <p>This routine deletes any zbuf segments in the specified zbuf, then deletes the zbuf ID itself. <i>zbufId</i> must not be used after this routine executes successfully.</p> <p>For any data buffers that were not in use by any other zbuf, <i>zbufDelete()</i> calls the associated free routine (callback).</p> |
| RETURNS | OK, or ERROR if the zbuf cannot be deleted. |
| SEE ALSO | <i>zbufLib</i> , <i>zbufCreate()</i> , <i>zbufInsertBuf()</i> |

zbufDup()

| | |
|--------------------|--|
| NAME | <i>zbufDup()</i> – duplicate a zbuf |
| SYNOPSIS | <pre>ZBUF_ID zbufDup (ZBUF_ID zbufId, /* zbuf to duplicate */ ZBUF_SEG zbufSeg, /* zbuf segment base for offset */ int offset, /* relative byte offset */ int len /* number of bytes to duplicate */)</pre> |
| DESCRIPTION | <p>This routine duplicates <i>len</i> bytes of <i>zbufId</i> starting at the specified byte location, and returns the zbuf ID of the newly created duplicate zbuf.</p> <p>The starting location of duplication is specified by <i>zbufSeg</i> and <i>offset</i>. See the <i>zbufLib</i> manual page for more information on specifying a byte location within a zbuf. In particular, the first byte duplicated is the exact byte specified by <i>zbufSeg</i> and <i>offset</i>.</p> <p>The number of bytes to duplicate is given by <i>len</i>. If this parameter is negative, or is larger than the number of bytes in the zbuf after the specified byte location, the rest of the zbuf is duplicated.</p> |

Duplication of zbuf data does not usually involve copying of the data. Instead, the zbuf segment pointer information is duplicated, while the data is not, which means that the data is shared among all zbuf segments that reference the data. See the **zbufLib** manual page for more information on copying and sharing zbuf data.

RETURNS The zbuf ID of a newly created duplicate zbuf, or NULL if the operation fails.

SEE ALSO **zbufLib**

zbufExtractCopy()

NAME *zbufExtractCopy()* – copy data from a zbuf to a buffer

SYNOPSIS

```
int zbufExtractCopy
(
    ZBUF_ID zbufId, /* zbuf from which data is copied */
    ZBUF_SEG zbufSeg, /* zbuf segment base for offset */
    int offset, /* relative byte offset */
    caddr_t buf, /* buffer into which data is copied */
    int len /* number of bytes to copy */
)
```

DESCRIPTION This routine copies *len* bytes of data from *zbufId* to the application buffer *buf*.

The starting location of the copy is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, the first byte copied is the exact byte specified by *zbufSeg* and *offset*.

The number of bytes to copy is given by *len*. If this parameter is negative, or is larger than the number of bytes in the zbuf after the specified byte location, the rest of the zbuf is copied. The bytes copied may span more than one segment.

RETURNS The number of bytes copied from the zbuf to the buffer, or ERROR if the operation fails.

SEE ALSO **zbufLib**

zbufInsert()

NAME *zbufInsert()* – insert a zbuf into another zbuf

SYNOPSIS

```
ZBUF_SEG zbufInsert
(
    ZBUF_ID  zbufId1, /* zbuf to insert zbufId2 into */
    ZBUF_SEG zbufSeg, /* zbuf segment base for offset */
    int      offset, /* relative byte offset */
    ZBUF_ID  zbufId2 /* zbuf to insert into zbufId1 */
)
```

DESCRIPTION

This routine inserts all *zbufId2* zbuf segments into *zbufId1* at the specified byte location. The location of insertion is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, insertion within a zbuf occurs before the byte location specified by *zbufSeg* and *offset*. Additionally, *zbufSeg* and *offset* must be NULL and 0, respectively, when inserting into an empty zbuf. After all the *zbufId2* segments are inserted into *zbufId1*, the zbuf ID *zbufId2* is deleted. *zbufId2* must not be used after this routine executes successfully.

RETURNS The zbuf segment ID for the first inserted segment, or NULL if the operation fails.

SEE ALSO *zbufLib*

zbufInsertBuf()

NAME *zbufInsertBuf()* – create a zbuf segment from a buffer and insert into a zbuf

SYNOPSIS

```
ZBUF_SEG zbufInsertBuf
(
    ZBUF_ID  zbufId, /* zbuf in which buffer is inserted */
    ZBUF_SEG zbufSeg, /* zbuf segment base for offset */
    int      offset, /* relative byte offset */
    caddr_t  buf,    /* application buffer for segment */
    int      len,    /* number of bytes to insert */
    VOIDFUNCPTR freeRtn, /* free-routine callback */
    int      freeArg /* argument to free routine */
)
```

zbufInsertCopy()

- DESCRIPTION** This routine creates a zbuf segment from the application buffer *buf* and inserts it at the specified byte location in *zbufId*.
- The location of insertion is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, insertion within a zbuf occurs before the byte location specified by *zbufSeg* and *offset*. Additionally, *zbufSeg* and *offset* must be NULL and 0, respectively, when inserting into an empty zbuf.
- The parameter *freeRtn* specifies a free-routine callback that runs when the data buffer *buf* is no longer referenced by any zbuf segments. If *freeRtn* is NULL, the zbuf functions normally, except that the application is not notified when no more zbufs segments reference *buf*. The free-routine callback runs from the context of the task that last deletes reference to the buffer. Declare the *freeRtn* callback as follows (using whatever routine name suits your application):
- ```
void freeCallback
(
 caddr_t buf, /* pointer to application buffer */
 int freeArg /* argument to free routine */
)
```
- RETURNS** The zbuf segment ID of the inserted segment, or NULL if the operation fails.
- SEE ALSO** [zbufLib](#)

---

## **zbufInsertCopy()**

- NAME** *zbufInsertCopy()* – copy buffer data into a zbuf
- SYNOPSIS**
- ```
ZBUF_SEG zbufInsertCopy
(
    ZBUF_ID zbufId, /* zbuf into which data is copied */
    ZBUF_SEG zbufSeg, /* zbuf segment base for offset */
    int offset, /* relative byte offset */
    caddr_t buf, /* buffer from which data is copied */
    int len /* number of bytes to copy */
)
```
- DESCRIPTION** This routine copies *len* bytes of data from the application buffer *buf* and inserts it at the specified byte location in *zbufId*. The application buffer is in no way tied to the zbuf after this operation; a separate copy of the data is made.
- The location of insertion is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, insertion

within a zbuf occurs before the byte location specified by *zbufSeg* and *offset*. Additionally, *zbufSeg* and *offset* must be NULL and 0, respectively, when inserting into an empty zbuf.

RETURNS The zbuf segment ID of the first inserted segment, or NULL if the operation fails.

SEE ALSO ***zbufLib***

zbufLength()

NAME *zbufLength()* – determine the length in bytes of a zbuf

SYNOPSIS

```
int zbufLength
(
    ZBUF_ID zbufId /* zbuf to determine length */
)
```

DESCRIPTION This routine returns the number of bytes in the zbuf *zbufId*.

RETURNS The number of bytes in the zbuf, or ERROR if the operation fails.

SEE ALSO ***zbufLib***

zbufSegData()

NAME *zbufSegData()* – determine the location of data in a zbuf segment

SYNOPSIS

```
caddr_t zbufSegData
(
    ZBUF_ID zbufId, /* zbuf to examine */
    ZBUF_SEG zbufSeg /* segment to get pointer to data */
)
```

DESCRIPTION This routine returns the location of the first byte of data in the zbuf segment *zbufSeg*. If *zbufSeg* is NULL, the location of data in the first segment in *zbufId* is returned.

RETURNS A pointer to the first byte of data in the specified zbuf segment, or NULL if the operation fails.

SEE ALSO ***zbufLib***

zbufSegFind()

zbufSegFind()

NAME *zbufSegFind()* – find the zbuf segment containing a specified byte location

SYNOPSIS

```
ZBUF_SEG zbufSegFind
(
    ZBUF_ID  zbufId, /* zbuf to examine */
    ZBUF_SEG zbufSeg, /* zbuf segment base for pOffset */
    int *    pOffset /* relative byte offset */
)
```

DESCRIPTION This routine translates an address within a zbuf to its most local formulation. *zbufSegFind()* locates the zbuf segment in *zbufId* that contains the byte location specified by *zbufSeg* and **pOffset*, then returns that zbuf segment, and writes in **pOffset* the new offset relative to the returned segment.

If the *zbufSeg*, **pOffset* pair specify a byte location past the end of the zbuf, or before the first byte in the zbuf, *zbufSegFind()* returns NULL.

See the **zbufLib** manual page for a full discussion of addressing zbufs by segment and offset.

RETURNS The zbuf segment ID of the segment containing the specified byte, or NULL if the operation fails.

SEE ALSO **zbufLib**

zbufSegLength()

NAME *zbufSegLength()* – determine the length of a zbuf segment

SYNOPSIS

```
int zbufSegLength
(
    ZBUF_ID  zbufId, /* zbuf to examine */
    ZBUF_SEG zbufSeg /* segment to determine length of */
)
```

DESCRIPTION This routine returns the number of bytes in the zbuf segment *zbufSeg*. If *zbufSeg* is NULL, the length of the first segment in *zbufId* is returned.

RETURNS The number of bytes in the specified zbuf segment, or ERROR if the operation fails.

SEE ALSO **zbufLib**

zbufSegNext()

NAME *zbufSegNext()* – get the next segment in a zbuf

SYNOPSIS

```
ZBUF_SEG zbufSegNext
(
    ZBUF_ID zbufId, /* zbuf to examine */
    ZBUF_SEG zbufSeg /* segment to get next segment */
)
```

DESCRIPTION This routine finds the zbuf segment in *zbufld* that is just after the zbuf segment *zbufSeg*. If *zbufSeg* is NULL, the segment after the first segment in *zbufld* is returned. If *zbufSeg* is the last segment in *zbufld*, NULL is returned.

RETURNS The zbuf segment ID of the segment after *zbufSeg*, or NULL if the operation fails.

SEE ALSO **zbufLib**

zbufSegPrev()

NAME *zbufSegPrev()* – get the previous segment in a zbuf

SYNOPSIS

```
ZBUF_SEG zbufSegPrev
(
    ZBUF_ID zbufId, /* zbuf to examine */
    ZBUF_SEG zbufSeg /* segment to get previous segment */
)
```

DESCRIPTION This routine finds the zbuf segment in *zbufld* that is just previous to the zbuf segment *zbufSeg*. If *zbufSeg* is NULL, or is the first segment in *zbufld*, NULL is returned.

RETURNS The zbuf segment ID of the segment previous to *zbufSeg*, or NULL if the operation fails.

SEE ALSO **zbufLib**

zbufSockBufSend()

NAME *zbufSockBufSend()* – create a zbuf from user data and send it to a TCP socket

SYNOPSIS

```
int zbufSockBufSend
(
    int          s,          /* socket to send to */
    char *      buf,        /* pointer to data buffer */
    int         bufLen,     /* number of bytes to send */
    VOIDFUNCPTR freeRtn,    /* free routine callback */
    int         freeArg,    /* argument to free routine */
    int         flags       /* flags to underlying protocols */
)
```

DESCRIPTION This routine creates a zbuf from the user buffer *buf*, and transmits it to a previously established connection-based (stream) socket.

The user-provided free routine callback at *freeRtn* is called when *buf* is no longer in use by the TCP/IP network stack. Applications can exploit this callback to receive notification that *buf* is free. If *freeRtn* is NULL, the routine functions normally, except that the application has no way of being notified when *buf* is released by the network stack. The free routine runs in the context of the task that last references the buffer. This is typically either the context of tNetTask, or the context of the caller's task. Declare *freeRtn* as follows (using whatever name is convenient):

```
void freeCallback
(
    caddr_t      buf,        /* pointer to user buffer */
    int         freeArg     /* user-provided argument to free routine */
)
```

You may OR the following values into the *flags* parameter with this operation:

MSG_OOB (0x1)

Out-of-band data.

MSG_DONTROUTE (0x4)

Send without using routing tables.

RETURNS The number of bytes sent, or ERROR if the call fails.

SEE ALSO *zbufSockLib*, *zbufSockSend()*, *send()*

zbufSockBufSendto()

NAME *zbufSockBufSendto()* – create a zbuf from a user message and send it to a UDP socket

SYNOPSIS

```
int zbufSockBufSendto
(
    int          s,          /* socket to send to */
    char *       buf,       /* pointer to data buffer */
    int          bufLen,    /* number of bytes to send */
    VOIDFUNCPTR freeRtn,    /* free routine callback */
    int          freeArg,   /* argument to free routine */
    int          flags,     /* flags to underlying protocols */
    struct sockaddr * to,   /* recipient's address */
    int          tolen     /* length of to socket addr */
)
```

DESCRIPTION This routine creates a zbuf from the user buffer *buf*, and sends it to the datagram socket named by *to*. The socket *s* is the sending socket.

The user-provided free routine callback at *freeRtn* is called when *buf* is no longer in use by the UDP/IP network stack. Applications can exploit this callback to receive notification that *buf* is free. If *freeRtn* is NULL, the routine functions normally, except that the application has no way of being notified when *buf* is released by the network stack. The free routine runs in the context of the task that last references the buffer. This is typically either tNetTask context, or the caller's task context. Declare *freeRtn* as follows (using whatever name is convenient):

```
void freeCallback
(
    caddr_t      buf,       /* pointer to user buffer */
    int          freeArg /* user-provided argument to free routine */
)
```

You may OR the following values into the *flags* parameter with this operation:

MSG_OOB (0x1)
Out-of-band data.

MSG_DONTROUTE (0x4)
Send without using routing tables.

RETURNS The number of bytes sent, or ERROR if the call fails.

SEE ALSO *zbufSockLib*, *zbufSockSendto()*, *sendto()*

zbufSockLibInit()

| | |
|--------------------|--|
| NAME | <i>zbufSockLibInit()</i> – initialize the zbuf socket interface library |
| SYNOPSIS | <pre>STATUS zbufSockLibInit (void)</pre> |
| DESCRIPTION | This routine initializes the zbuf socket interface library. It must be called before any zbuf socket routines are used. It is called automatically when the configuration macro <code>INCLUDE_ZBUF_SOCKET</code> is defined. |
| RETURNS | OK, or ERROR if the zbuf socket interface could not be initialized. |
| SEE ALSO | <i>zbufSockLib</i> |

zbufSockRecv()

| | |
|--------------------|--|
| NAME | <i>zbufSockRecv()</i> – receive data in a zbuf from a TCP socket |
| SYNOPSIS | <pre>ZBUF_ID zbufSockRecv (int s, /* socket to receive data from */ int flags, /* flags to underlying protocols */ int * pLen /* number of bytes requested/returned */)</pre> |
| DESCRIPTION | <p>This routine receives data from a connection-based (stream) socket, and returns the data to the user in a newly created zbuf.</p> <p>The <i>pLen</i> parameter indicates the number of bytes requested by the caller. If the operation is successful, the number of bytes received is copied to <i>pLen</i>.</p> <p>You may OR the following values into the <i>flags</i> parameter with this operation:</p> <ul style="list-style-type: none">MSG_OOB (0x1) Out-of-band data.MSG_PEEK (0x2) Return data without removing it from socket. <p>Once the user application is finished with the zbuf, <i>zbufDelete()</i> should be called to return the zbuf memory buffer to the VxWorks network stack.</p> |

RETURNS The zbuf ID of a newly created zbuf containing the received data, or NULL if the operation fails.

SEE ALSO `zbufSockLib`, `recv()`

zbufSockRecvfrom()

NAME `zbufSockRecvfrom()` – receive a message in a zbuf from a UDP socket

SYNOPSIS

```
ZBUF_ID zbufSockRecvfrom
(
    int          s,          /* socket to receive from */
    int          flags,     /* flags to underlying protocols */
    int *        pLen,      /* number of bytes requested/returned */
    struct sockaddr * from, /* where to copy sender's addr */
    int *        pFromLen /* value/result length of from */
)
```

DESCRIPTION This routine receives a message from a datagram socket, and returns the message to the user in a newly created zbuf.

The message is received regardless of whether the socket is connected. If *from* is nonzero, the address of the sender's socket is copied to it. Initialize the value-result parameter *pFromLen* to the size of the *from* buffer. On return, *pFromLen* contains the actual size of the address stored in *from*.

The *pLen* parameter indicates the number of bytes requested by the caller. If the operation is successful, the number of bytes received is copied to *pLen*.

You may OR the following values into the *flags* parameter with this operation:

MSG_OOB (0x1)
Out-of-band data.

MSG_PEEK (0x2)
Return data without removing it from socket.

Once the user application is finished with the zbuf, `zbufDelete()` should be called to return the zbuf memory buffer to the VxWorks network stack.

RETURNS The zbuf ID of a newly created zbuf containing the received message, or NULL if the operation fails.

SEE ALSO `zbufSockLib`

zbufSockSend()

NAME *zbufSockSend()* – send zbuf data to a TCP socket

SYNOPSIS

```
int zbufSockSend
(
    int      s,          /* socket to send to */
    ZBUF_ID zbufId,     /* zbuf to transmit */
    int      zbufLen,   /* length of entire zbuf */
    int      flags      /* flags to underlying protocols */
)
```

DESCRIPTION This routine transmits all of the data in *zbufId* to a previously established connection-based (stream) socket.

The *zbufLen* parameter is used only for determining the amount of space needed from the socket write buffer. *zbufLen* has no effect on how many bytes are sent; the entire zbuf is always transmitted. If the length of *zbufId* is not known, the caller must first determine it by calling *zbufLength()*.

This routine transfers ownership of the zbuf from the user application to the VxWorks network stack. The zbuf ID *zbufId* is deleted by this routine, and should not be used after the routine is called, even if an ERROR status is returned. (Exceptions: when the routine fails because the zbuf socket interface library was not initialized or an invalid zbuf ID was passed in, in which case there is no zbuf to delete. Moreover, if the call fails during a non-blocking I/O socket write with an **errno** of **EWOULDBLOCK**, then *zbufId* is not deleted; thus the caller may send it again at a later time.)

You may OR the following values into the *flags* parameter with this operation:

MSG_OOB (0x1)

Out-of-band data.

MSG_DONTROUTE (0x4)

Send without using routing tables.

RETURNS The number of bytes sent, or ERROR if the call fails.

SEE ALSO *zbufSockLib*, *zbufLength()*, *zbufSockBufSend()*, *send()*

zbufSockSendto()

NAME *zbufSockSendto()* – send a zbuf message to a UDP socket

SYNOPSIS

```
int zbufSockSendto
(
    int          s,          /* socket to send to */
    ZBUF_ID     zbufId,    /* zbuf to transmit */
    int         zbufLen,    /* length of entire zbuf */
    int         flags,      /* flags to underlying protocols */
    struct sockaddr * to,   /* recipient's address */
    int         tolen       /* length of to socket addr */
)
```

DESCRIPTION This routine sends the entire message in *zbufId* to the datagram socket named by *to*. The socket *s* is the sending socket.

The *zbufLen* parameter is used only for determining the amount of space needed from the socket write buffer. *zbufLen* has no effect on how many bytes are sent; the entire zbuf is always transmitted. If the length of *zbufId* is not known, the caller must first determine it by calling *zbufLength()*.

This routine transfers ownership of the zbuf from the user application to the VxWorks network stack. The zbuf ID *zbufId* is deleted by this routine, and should not be used after the routine is called, even if an ERROR status is returned. (Exceptions: when the routine fails because the zbuf socket interface library was not initialized or an invalid zbuf ID was passed in, in which case there is no zbuf to delete. Moreover, if the call fails during a non-blocking I/O socket write with an **errno** of **EWOULDBLOCK**, then *zbufId* is not deleted; thus the caller may send it again at a later time.)

You may OR the following values into the *flags* parameter with this operation:

MSG_OOB (0x1)
Out-of-band data.

MSG_DONTROUTE (0x4)
Send without using routing tables.

RETURNS The number of bytes sent, or ERROR if the call fails.

SEE ALSO *zbufSockLib*, *zbufLength()*, *zbufSockBufSendto()*, *sendto()*

zbufSplit()

NAME *zbufSplit()* – split a zbuf into two separate zbufs

SYNOPSIS

```
ZBUF_ID zbufSplit
(
    ZBUF_ID zbufId, /* zbuf to split into two */
    ZBUF_SEG zbufSeg, /* zbuf segment base for offset */
    int offset /* relative byte offset */
)
```

DESCRIPTION This routine splits *zbufId* into two separate zbufs at the specified byte location. The first portion remains in *zbufId*, while the end portion is returned in a newly created zbuf. The location of the split is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, after the split operation, the first byte of the returned zbuf is the exact byte specified by *zbufSeg* and *offset*.

RETURNS The zbuf ID of a newly created zbuf containing the end portion of *zbufId*, or NULL if the operation fails.

SEE ALSO **zbufLib**

Keyword Index

| | Keyword | Name | Page |
|--|------------------------------|---|-------|
| | initialize NS | 16550 chip. <i>evbNs16550HrdInit()</i> | 2-182 |
| | /interrupt for NS | 16550 chip. <i>evbNs16550Int()</i> | 2-183 |
| | NS | 16550 UART tty driver. <i>ns16550Sio</i> | 1-273 |
| | interface driver for 3COM | 3C509. END network <i>elt3c509End</i> | 1-104 |
| | display statistics for | 3C509 elt network interface. <i>eltShow()</i> | 2-163 |
| | interface driver. 3Com | 3C509 Ethernet network <i>if_elt</i> | 1-144 |
| | registers for NCR | 53C710. /hardware-dependent <i>ncr710SetHwRegisterScsi2()</i> | 2-490 |
| | (SIOP) library (SCSI-1). NCR | 53C710 SCSI I/O Processor <i>ncr710Lib</i> | 1-255 |
| | (SIOP) library (SCSI-2). NCR | 53C710 SCSI I/O Processor <i>ncr710Lib2</i> | 1-256 |
| | control structure for NCR | 53C710 SIOP. create <i>ncr710CtrlCreate()</i> | 2-485 |
| | control structure for NCR | 53C710 SIOP. create <i>ncr710CtrlCreateScsi2()</i> | 2-486 |
| | control structure for NCR | 53C710 SIOP. initialize <i>ncr710CtrlInit()</i> | 2-487 |
| | control structure for NCR | 53C710 SIOP. initialize <i>ncr710CtrlInitScsi2()</i> | 2-488 |
| | /registers for NCR | 53C710 SIOP. <i>ncr710SetHwRegister()</i> | 2-489 |
| | /values of all readable NCR | 53C710 SIOP registers. <i>ncr710Show()</i> | 2-491 |
| | /values of all readable NCR | 53C710 SIOP registers. <i>ncr710ShowScsi2()</i> | 2-492 |
| | (SIOP) library (SCSI-2). NCR | 53C8xx PCI SCSI I/O Processor <i>ncr810Lib</i> | 1-257 |
| | control structure for NCR | 53C8xx SIOP. create <i>ncr810CtrlCreatel()</i> | 2-493 |
| | control structure for NCR | 53C8xx SIOP. initialize <i>ncr810CtrlInit()</i> | 2-494 |
| | /registers for NCR | 53C8xx SIOP. <i>ncr810SetHwRegister()</i> | 2-495 |
| | /values of all readable NCR | 53C8xx SIOP registers. <i>ncr810Show()</i> | 2-496 |
| | (ASC) library (SCSI-1). NCR | 53C90 Advanced SCSI Controller <i>ncr5390Lib1</i> | 1-258 |
| | (ASC) library (SCSI-2). NCR | 53C90 Advanced SCSI Controller <i>ncr5390Lib2</i> | 1-259 |
| | control structure for NCR | 53C90 ASC. create <i>ncr5390CtrlCreatel()</i> | 2-497 |
| | control structure for NCR | 53C90 ASC. create <i>ncr5390CtrlCreateScsi2()</i> | 2-498 |
| | driver. Motorola | 68EN302 network-interface <i>if_mbc</i> | 1-160 |
| | clear entry from cache | (68K, x86). <i>cacheArchClearEntry()</i> | 2-48 |
| | display statistics for SMC | 8013WC elc network interface. <i>elcShow()</i> | 2-161 |
| | interface driver. SMC | 8013WC Ethernet network <i>if_elc</i> | 1-144 |
| | adaptor chip library. Intel | 82365SL PCMCIA host bus <i>pcic</i> | 1-280 |
| | adaptor chip show/ Intel | 82365SL PCMCIA host bus <i>pcicShow</i> | 1-281 |

| | Keyword | Name | Page |
|---|--------------------------------------|-----------------------------------|--------|
| interface/ END style Intel | 82557 Ethernet network | fei82557End | 1-114 |
| interface driver. Intel | 82557 Ethernet network | if_fei | 1-149 |
| for handling interrupts from | 82596. entry point | eiInt() | 2-156 |
| interface driver. Intel | 82596 Ethernet network | if_ei | 1-137 |
| interface/ END style Intel | 82596 Ethernet network | ei82596End | 1-96 |
| interface driver for/ Intel | 82596 Ethernet network | if_eihk | 1-140 |
| return contents of register | a0 (also a1 - a7) (MC680x0). | a0() | 2-1 |
| change | abort character. | tyAbortSet() | 2-927 |
| set | abort function. | tyAbortFuncSet() | 2-927 |
| compute | absolute value (ANSI). | fabs() | 2-191 |
| compute | absolute value (ANSI). | fabsf() | 2-192 |
| (ANSI). compute | absolute value of integer | abs() | 2-2 |
| compute | absolute value of long (ANSI). | labs() | 2-313 |
| | accept connection from socket. | accept() | 2-2 |
| filtering. | activate strict border gateway | ripFilterEnable() | 2-637 |
| | activate task. | VXWTask::activate() | 2-1027 |
| initialized. | activate task that has been | taskActivate() | 2-865 |
| initialize asynchronous I/O | (AIO) library. | aioPxLibInit() | 2-8 |
| asynchronous I/O | (AIO) library (POSIX). | aioPxLib | 1-4 |
| show | AIO requests. | aioShow() | 2-8 |
| asynchronous I/O | (AIO) show library. | aioPxShow | 1-8 |
| | AIO system driver. | aioSysDrv | 1-9 |
| initialize | AIO system driver. | aioSysInit() | 2-9 |
| | allocate aligned memory. | memalign() | 2-415 |
| partition. | allocate aligned memory from | memPartAlignedAlloc() | 2-422 |
| partition. | allocate aligned memory from | VXWMemPart::alignedAlloc() | 2-988 |
| partition. | allocate block of memory from | memPartAlloc() | 2-423 |
| partition. | allocate block of memory from | VXWMemPart::alloc() | 2-989 |
| shared memory system/ system memory partition/ DMA devices and drivers. | allocate block of memory from | smMemMalloc() | 2-762 |
| shared memory system/ boundary. | allocate block of memory from | malloc() | 2-396 |
| (ANSI). | allocate cache-safe buffer for | cacheDmaMalloc() | 2-54 |
| clock for timing base/ network interface/ END style | allocate memory for array from | smMemCalloc() | 2-760 |
| network interface driver. | allocate memory on page | valloc() | 2-955 |
| Ethernet network interface/ Ethernet driver. END style | allocate space for array | calloc() | 2-77 |
| abnormal program termination | allocate timer using specified | timer_create() | 2-912 |
| absolute value of integer | AMD 7990 LANCE Ethernet | ln7990End | 1-190 |
| compute arc cosine | AMD Am7990 LANCE Ethernet | if_ln | 1-153 |
| compute arc cosine | AMD Am79C970 PCnet-PCI | if_lnPci | 1-156 |
| broken-down time into string | AMD Am79C97X PCnet-PCI | ln97xEnd | 1-186 |
| compute arc sine | (ANSI). cause | abort() | 2-1 |
| compute arc sine | (ANSI). compute | abs() | 2-2 |
| put diagnostics into programs | (ANSI). | acos() | 2-3 |
| compute arc tangent | (ANSI). | acosf() | 2-3 |
| compute arc tangent of y/x | (ANSI). convert | asctime() | 2-18 |
| compute arc tangent of y/x | (ANSI). | asin() | 2-19 |
| | (ANSI). | asinf() | 2-19 |
| | (ANSI). | assert() | 2-20 |
| | (ANSI). | atan() | 2-22 |
| | (ANSI). | atan2() | 2-22 |
| | (ANSI). | atan2f() | 2-23 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|-------------------|-------|
| compute arc tangent | (ANSI). | <i>atanf()</i> | 2-23 |
| termination (Unimplemented) | (ANSI). /function at program | <i>atexit()</i> | 2-25 |
| convert string to double | (ANSI). | <i>atof()</i> | 2-26 |
| convert string to int | (ANSI). | <i>atoi()</i> | 2-26 |
| convert string to long | (ANSI). | <i>atol()</i> | 2-27 |
| perform binary search | (ANSI). | <i>bsearch()</i> | 2-45 |
| allocate space for array | (ANSI). | <i>calloc()</i> | 2-77 |
| or equal to specified value | (ANSI). /integer greater than | <i>ceil()</i> | 2-82 |
| or equal to specified value | (ANSI). /integer greater than | <i>ceilf()</i> | 2-83 |
| and error flags for stream | (ANSI). clear end-of-file | <i>clearerr()</i> | 2-88 |
| processor time in use | (ANSI). determine | <i>clock()</i> | 2-88 |
| compute cosine | (ANSI). | <i>cos()</i> | 2-94 |
| compute cosine | (ANSI). | <i>cosf()</i> | 2-95 |
| compute hyperbolic cosine | (ANSI). | <i>cosh()</i> | 2-95 |
| compute hyperbolic cosine | (ANSI). | <i>coshf()</i> | 2-96 |
| time in seconds into string | (ANSI). convert | <i>ctime()</i> | 2-106 |
| between two calendar times | (ANSI). compute difference | <i>difftime()</i> | 2-136 |
| compute quotient and remainder | (ANSI). | <i>div()</i> | 2-137 |
| exit task | (ANSI). | <i>exit()</i> | 2-190 |
| compute exponential value | (ANSI). | <i>exp()</i> | 2-190 |
| compute exponential value | (ANSI). | <i>expf()</i> | 2-191 |
| compute absolute value | (ANSI). | <i>fabs()</i> | 2-191 |
| compute absolute value | (ANSI). | <i>fabsf()</i> | 2-192 |
| close stream | (ANSI). | <i>fclose()</i> | 2-192 |
| indicator for stream | (ANSI). test end-of-file | <i>feof()</i> | 2-198 |
| indicator for file pointer | (ANSI). test error | <i>ferror()</i> | 2-199 |
| flush stream | (ANSI). | <i>fflush()</i> | 2-199 |
| next character from stream | (ANSI). return | <i>fgetc()</i> | 2-200 |
| position indicator for stream | (ANSI). /current value of file | <i>fgetpos()</i> | 2-200 |
| of characters from stream | (ANSI). read specified number | <i>fgets()</i> | 2-201 |
| or equal to specified value | (ANSI). /integer less than | <i>floor()</i> | 2-205 |
| or equal to specified value | (ANSI). /integer less than | <i>floorf()</i> | 2-205 |
| compute remainder of x/y | (ANSI). | <i>fmod()</i> | 2-206 |
| compute remainder of x/y | (ANSI). | <i>fmodf()</i> | 2-206 |
| open file specified by name | (ANSI). | <i>fopen()</i> | 2-207 |
| formatted string to stream | (ANSI). write | <i>fprintf()</i> | 2-215 |
| write character to stream | (ANSI). | <i>fputc()</i> | 2-219 |
| write string to stream | (ANSI). | <i>fputs()</i> | 2-219 |
| read data into array | (ANSI). | <i>fread()</i> | 2-220 |
| free block of memory | (ANSI). | <i>free()</i> | 2-220 |
| open file specified by name | (ANSI). | <i>freopen()</i> | 2-221 |
| fraction and power of 2 | (ANSI). /into normalized | <i>frexp()</i> | 2-221 |
| convert characters from stream | (ANSI). read and | <i>fscanf()</i> | 2-222 |
| position indicator for stream | (ANSI). set file | <i>fseek()</i> | 2-226 |
| position indicator for stream | (ANSI). set file | <i>fsetpos()</i> | 2-227 |
| position indicator for stream | (ANSI). /current value of file | <i>ftell()</i> | 2-229 |
| write from specified array | (ANSI). | <i>fwrite()</i> | 2-238 |
| next character from stream | (ANSI). return | <i>getc()</i> | 2-239 |
| from standard input stream | (ANSI). return next character | <i>getchar()</i> | 2-239 |
| get environment variable | (ANSI). | <i>getenv()</i> | 2-240 |

| | Keyword | Name | Page |
|----------------------------------|--------------------------------------|---------------------|-------|
| from standard input stream | (ANSI). read characters | <i>gets()</i> | 2-242 |
| time into UTC broken-down time | (ANSI). convert calendar | <i>gmtime()</i> | 2-245 |
| character is alphanumeric | (ANSI). test whether | <i>isalnum()</i> | 2-304 |
| whether character is letter | (ANSI). test | <i>isalpha()</i> | 2-304 |
| character is control character | (ANSI). test whether | <i>isctrnl()</i> | 2-305 |
| character is decimal digit | (ANSI). test whether | <i>isdigit()</i> | 2-306 |
| non-white-space character | (ANSI). /is printing, | <i>isgraph()</i> | 2-306 |
| character is lower-case letter | (ANSI). test whether | <i>islower()</i> | 2-307 |
| including space character | (ANSI). /is printable, | <i>isprint()</i> | 2-307 |
| character is punctuation | (ANSI). test whether | <i>ispunct()</i> | 2-308 |
| is white-space character | (ANSI). /whether character | <i>isspace()</i> | 2-308 |
| character is upper-case letter | (ANSI). test whether | <i>isupper()</i> | 2-309 |
| character is hexadecimal digit | (ANSI). test whether | <i>isxdigit()</i> | 2-309 |
| compute absolute value of long | (ANSI). | <i>labs()</i> | 2-313 |
| number by integral power of 2 | (ANSI). multiply | <i>ldexp()</i> | 2-314 |
| and remainder of division | (ANSI). compute quotient | <i>ldiv()</i> | 2-315 |
| of object with type <i>lconv</i> | (ANSI). set components | <i>localeconv()</i> | 2-330 |
| time into broken-down time | (ANSI). convert calendar | <i>localtime()</i> | 2-333 |
| compute natural logarithm | (ANSI). | <i>log()</i> | 2-334 |
| compute base-10 logarithm | (ANSI). | <i>log10()</i> | 2-335 |
| compute base-10 logarithm | (ANSI). | <i>log10f()</i> | 2-336 |
| compute natural logarithm | (ANSI). | <i>logf()</i> | 2-336 |
| by restoring saved environment | (ANSI). /non-local goto | <i>longjmp()</i> | 2-346 |
| from system memory partition | (ANSI). /block of memory | <i>malloc()</i> | 2-396 |
| character (Unimplemented) | (ANSI). /length of multibyte | <i>mblen()</i> | 2-413 |
| to wide chars (Unimplemented) | (ANSI). /of multibyte chars | <i>mbstowcs()</i> | 2-413 |
| wide character (Unimplemented) | (ANSI). /character to | <i>mbtowc()</i> | 2-414 |
| block of memory for character | (ANSI). search | <i>memchr()</i> | 2-416 |
| compare two blocks of memory | (ANSI). | <i>memcmp()</i> | 2-416 |
| from one location to another | (ANSI). copy memory | <i>memcpy()</i> | 2-417 |
| from one location to another | (ANSI). copy memory | <i>memmove()</i> | 2-421 |
| set block of memory | (ANSI). | <i>memset()</i> | 2-428 |
| time into calendar time | (ANSI). convert broken-down | <i>mktime()</i> | 2-432 |
| integer and fraction parts | (ANSI). /number into | <i>modf()</i> | 2-435 |
| in <i>errno</i> to error message | (ANSI). map error number | <i>perror()</i> | 2-569 |
| raised to specified power | (ANSI). /value of number | <i>pow()</i> | 2-572 |
| raised to specified power | (ANSI). /value of number | <i>powf()</i> | 2-573 |
| to standard output stream | (ANSI). /formatted string | <i>printf()</i> | 2-591 |
| write character to stream | (ANSI). | <i>putc()</i> | 2-603 |
| to standard output stream | (ANSI). write character | <i>putchar()</i> | 2-603 |
| to standard output stream | (ANSI). write string | <i>puts()</i> | 2-604 |
| sort array of objects | (ANSI). | <i>qsort()</i> | 2-606 |
| between 0 and <i>RAND_MAX</i> | (ANSI). /pseudo-random integer | <i>rand()</i> | 2-610 |
| reallocate block of memory | (ANSI). | <i>realloc()</i> | 2-615 |
| remove file | (ANSI). | <i>remove()</i> | 2-621 |
| indicator to beginning of file | (ANSI). set file position | <i>rewind()</i> | 2-631 |
| from standard input stream | (ANSI). /convert characters | <i>scanf()</i> | 2-666 |
| specify buffering for stream | (ANSI). | <i>setbuf()</i> | 2-728 |
| in <i>jmp_buf</i> argument | (ANSI). /calling environment | <i>setjmp()</i> | 2-730 |
| set appropriate locale | (ANSI). | <i>setlocale()</i> | 2-731 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|-------------------|--------|
| specify buffering for stream | (ANSI). | <i>setobuf()</i> | 2-737 |
| compute sine | (ANSI). | <i>sin()</i> | 2-753 |
| compute sine | (ANSI). | <i>sinf()</i> | 2-754 |
| compute hyperbolic sine | (ANSI). | <i>sinh()</i> | 2-755 |
| compute hyperbolic sine | (ANSI). | <i>sinhf()</i> | 2-755 |
| formatted string to buffer | (ANSI). write | <i>sprintf()</i> | 2-800 |
| non-negative square root | (ANSI). compute | <i>sqrt()</i> | 2-804 |
| non-negative square root | (ANSI). compute | <i>sqrtf()</i> | 2-805 |
| to generate random numbers | (ANSI). /value of seed used | <i>srand()</i> | 2-808 |
| characters from ASCII string | (ANSI). read and convert | <i>sscanf()</i> | 2-808 |
| one string to another | (ANSI). concatenate | <i>strcat()</i> | 2-818 |
| of character in string | (ANSI). find first occurrence | <i>strchr()</i> | 2-818 |
| two strings lexicographically | (ANSI). compare | <i>strcmp()</i> | 2-819 |
| as appropriate to LC_COLLATE | (ANSI). compare two strings | <i>strcoll()</i> | 2-819 |
| copy one string to another | (ANSI). | <i>strcpy()</i> | 2-820 |
| first character from given set | (ANSI). /string length up to | <i>strcspn()</i> | 2-820 |
| error number to error string | (ANSI). map | <i>strerror()</i> | 2-821 |
| time into formatted string | (ANSI). convert broken-down | <i>strftime()</i> | 2-822 |
| determine length of string | (ANSI). | <i>strlen()</i> | 2-823 |
| from one string to another | (ANSI). /characters | <i>strncat()</i> | 2-824 |
| n characters of two strings | (ANSI). compare first | <i>strncmp()</i> | 2-824 |
| from one string to another | (ANSI). copy characters | <i>strncpy()</i> | 2-825 |
| of character from given set | (ANSI). /occurrence in string | <i>strpbrk()</i> | 2-825 |
| of character in string | (ANSI). find last occurrence | <i>strrchr()</i> | 2-826 |
| character not in given set | (ANSI). /length up to first | <i>strspn()</i> | 2-826 |
| of substring in string | (ANSI). find first occurrence | <i>strstr()</i> | 2-827 |
| portion of string to double | (ANSI). convert initial | <i>strtod()</i> | 2-827 |
| break down string into tokens | (ANSI). | <i>strtok()</i> | 2-828 |
| convert string to long integer | (ANSI). | <i>strtol()</i> | 2-830 |
| to unsigned long integer | (ANSI). convert string | <i>strtoul()</i> | 2-831 |
| to n characters of s2 into s1 | (ANSI). transform up | <i>strxfrm()</i> | 2-833 |
| processor (Unimplemented) | (ANSI). /string to command | <i>system()</i> | 2-859 |
| compute tangent | (ANSI). | <i>tan()</i> | 2-860 |
| compute tangent | (ANSI). | <i>tanf()</i> | 2-861 |
| compute hyperbolic tangent | (ANSI). | <i>tanh()</i> | 2-861 |
| compute hyperbolic tangent | (ANSI). | <i>tanhf()</i> | 2-862 |
| current calendar time | (ANSI). determine | <i>time()</i> | 2-910 |
| binary file (Unimplemented) | (ANSI). create temporary | <i>tmpfile()</i> | 2-920 |
| generate temporary file name | (ANSI). | <i>tmpnam()</i> | 2-921 |
| to lower-case equivalent | (ANSI). /upper-case letter | <i>tolower()</i> | 2-921 |
| to upper-case equivalent | (ANSI). /lower-case letter | <i>toupper()</i> | 2-922 |
| back into input stream | (ANSI). push character | <i>ungetc()</i> | 2-940 |
| formatted string to stream | (ANSI). write | <i>vfprintf()</i> | 2-956 |
| list to standard output | (ANSI). /variable argument | <i>vprintf()</i> | 2-971 |
| argument list to buffer | (ANSI). /with variable | <i>vsprintf()</i> | 2-972 |
| chars (Unimplemented) | (ANSI). /chars to multibyte | <i>wcstombs()</i> | 2-1049 |
| character (Unimplemented) | (ANSI). /to multibyte | <i>wctomb()</i> | 2-1049 |
| | ANSI assert documentation. | ansiAssert | 1-12 |
| | ANSI ctype documentation. | ansiCtype | 1-12 |
| | ANSI locale documentation. | ansiLocale | 1-13 |

| | Keyword | Name | Page |
|---|---|-------------------|-------|
| | ANSI math documentation. | ansiMath | 1-13 |
| | ANSI setjmp documentation. | ansiSetjmp | 1-15 |
| | ANSI stdarg documentation. | ansiStdarg | 1-15 |
| | ANSI stdio documentation. | ansiStdio | 1-16 |
| | ANSI stdlib documentation. | ansiStdlib | 1-20 |
| | ANSI string documentation. | ansiString | 1-21 |
| | ANSI time documentation. | ansiTime | 1-22 |
| compute | arc cosine (ANSI). | acos() | 2-3 |
| compute | arc cosine (ANSI). | acosf() | 2-3 |
| compute | arc sine (ANSI). | asin() | 2-19 |
| compute | arc sine (ANSI). | asinf() | 2-19 |
| compute | arc tangent (ANSI). | atan() | 2-22 |
| compute | arc tangent (ANSI). | atanf() | 2-23 |
| compute | arc tangent of y/x (ANSI). | atan2() | 2-22 |
| compute | arc tangent of y/x (ANSI). | atan2f() | 2-23 |
| processor status register | (ARM). /contents of current | cpsr() | 2-103 |
| exception vector (PowerPC, ARM). /routine to asynchronous | exclIntConnect() | 2-186 | |
| CPU exception vector (PowerPC, ARM). get | excVecGet() | 2-188 | |
| CPU exception vector (PowerPC, ARM). set | excVecSet() | 2-189 | |
| interrupt bits (MIPS, PowerPC, ARM). disable corresponding | intDisable() | 2-278 | |
| interrupt bits (MIPS, PowerPC, ARM). enable corresponding | intEnable() | 2-278 | |
| (MC680x0, SPARC, i960, x86, ARM). set interrupt level | intLevelSet() | 2-279 | |
| (MC680x0, SPARC, i960, x86, ARM). /lock-out level | intLockLevelGet() | 2-282 | |
| (MC680x0, SPARC, i960, x86, ARM). /lock-out level | intLockLevelSet() | 2-282 | |
| uninitialized vector handler (ARM). set | intUninitVecSet() | 2-283 | |
| SPARC, i960, x86, MIPS, ARM). /base address (MC680x0, | intVecBaseGet() | 2-284 | |
| SPARC, i960, x86, MIPS, ARM). /base address (MC680x0, | intVecBaseSet() | 2-285 | |
| (MC680x0, SPARC, i960, x86, ARM). /exception vector table | intVecTableWriteProtect() | 2-288 | |
| PSR value, symbolically (ARM). /meaning of specified | psrShow() | 2-601 | |
| of register r0 (also r1 - r14) (ARM). return contents | r0() | 2-606 | |
| | ARM AMBA UART tty driver. | ambaSio | 1-9 |
| time until next expiration and initialize proxy | arm timer (POSIX). set timer_settime() | 2-914 | |
| Address Resolution Protocol (ARP) client library. proxy | proxyArpLibInit() | 2-595 | |
| display known ARP entries. | arptabShow() | 2-17 | |
| add, modify, or delete MIB-II ARP entry. | m2IpAtransTblEntrySet() | 2-365 | |
| Address Resolution Protocol (ARP) library. proxy | proxyArpLib | 1-296 | |
| create proxy ARP network. | proxyNetCreate() | 2-596 | |
| show proxy ARP networks. | proxyNetShow() | 2-597 | |
| add entry to system ARP table. | arpAdd() | 2-15 | |
| delete entry from system ARP table. | arpDelete() | 2-16 | |
| flush all entries in system ARP table. | arpFlush() | 2-16 | |
| display entries in system ARP table. | arpShow() | 2-17 | |
| get MIB-II ARP table entry. | m2IpAtransTblEntryGet() | 2-364 | |
| Address Resolution Protocol (ARP) table manipulation/ | arpLib | 1-24 | |
| read data into array (ANSI). | fread() | 2-220 | |
| write from specified array (ANSI). | fwrite() | 2-238 | |
| allocate space for array (ANSI). | calloc() | 2-77 | |
| system/ allocate memory for array from shared memory | smMemCalloc() | 2-760 | |
| sort array of objects (ANSI). | qsort() | 2-606 | |

| | Keyword | Name | Page |
|------------------|-----------------------------|--------------------------------------|---------------------------------------|
| | structure for NCR 53C90 | ASC. create control | <i>ncr5390CtrlCreate()</i> 2-497 |
| | structure for NCR 53C90 | ASC. create control | <i>ncr5390CtrlCreateScsi2()</i> 2-498 |
| 53C90 | Advanced SCSI Controller | (ASC) library (SCSI-1). NCR | <i>ncr5390Lib1</i> 1-258 |
| 53C90 | Advanced SCSI Controller | (ASC) library (SCSI-2). NCR | <i>ncr5390Lib2</i> 1-259 |
| | user-specified fields in | ASC structure. initialize | <i>ncr5390CtrlInit()</i> 2-499 |
| | and convert characters from | ASCII string (ANSI). read | <i>sscansf()</i> 2-808 |
| | (SPARC). probe address in | ASI space for bus error | <i>vxMemProbeAsi()</i> 2-974 |
| | /manipulation library SPARC | assembly language routines. | <i>bALib</i> 1-27 |
| | I960Cx cache management | assembly routines. | <i>cacheI960CxALib</i> 1-35 |
| | I960Jx cache management | assembly routines. | <i>cacheI960JxALib</i> 1-36 |
| MIPS | R3000 cache management | assembly routines. | <i>cacheR3kALib</i> 1-47 |
| | ANSI | assert documentation. | <i>ansiAssert</i> 1-12 |
| (Western Digital | WD33C93/ | assert RST line on SCSI bus | <i>sysScsiBusReset()</i> 2-855 |
| | connect C routine to | asynchronous exception vector/ | <i>exclntConnect()</i> 2-186 |
| | synchronization (POSIX). | asynchronous file | <i>aio_fsync()</i> 2-10 |
| | library. initialize | asynchronous I/O (AIO) | <i>aioPxLibInit()</i> 2-8 |
| | (POSIX). | asynchronous I/O (AIO) library | <i>aioPxLib</i> 1-4 |
| | library. | asynchronous I/O (AIO) show | <i>aioPxShow</i> 1-8 |
| | retrieve return status of | asynchronous I/O operation/ | <i>aio_return()</i> 2-11 |
| | retrieve error status of | asynchronous I/O operation/ | <i>aio_error()</i> 2-9 |
| | (POSIX). wait for | asynchronous I/O request(s) | <i>aio_suspend()</i> 2-12 |
| | (POSIX). initiate list of | asynchronous I/O requests | <i>lio_listio()</i> 2-318 |
| | processing routine for/ | asynchronous message | <i>snmpSaHandlerAsync()</i> 2-788 |
| | initiate | asynchronous read (POSIX). | <i>aio_read()</i> 2-11 |
| | initiate | asynchronous write (POSIX). | <i>aio_write()</i> 2-13 |
| | initialize | ATA driver. | <i>ataDrv()</i> 2-21 |
| mount | DOS file system from | ATA hard disk. | <i>usrAtaConfig()</i> 2-947 |
| | | ATA initialization. | <i>usrAta</i> 1-409 |
| | create device for | ATA/IDE disk. | <i>ataDevCreate()</i> 2-20 |
| | routine. initialize | ATA/IDE disk driver show | <i>ataShowInit()</i> 2-25 |
| | show | ATA/IDE disk parameters. | <i>ataShow()</i> 2-24 |
| | disk device driver. | ATA/IDE (LOCAL and PCMCIA) | <i>ataDrv</i> 1-24 |
| | disk device driver show/ | ATA/IDE (LOCAL and PCMCIA) | <i>ataShow</i> 1-27 |
| memory | objects facility (VxMP/ | attach calling CPU to shared | <i>smObjAttach()</i> 2-772 |
| | network stack. generic | attach routine for TCP/IP | <i>ipAttach()</i> 2-299 |
| | interface. | attach shared memory network | <i>smNetAttach()</i> 2-769 |
| | list of network interfaces/ | attach ULIP interface to | <i>ulattach()</i> 2-934 |
| | sample | authentication hook. | <i>ripAuthHook()</i> 2-633 |
| | interface. remove | authentication hook from RIP | <i>ripAuthHookDelete()</i> 2-636 |
| | interface. add | authentication hook to RIP | <i>ripAuthHookAdd()</i> 2-633 |
| | get NFS UNIX | authentication parameters. | <i>nfsAuthUnixGet()</i> 2-522 |
| | modify NFS UNIX | authentication parameters. | <i>nfsAuthUnixPrompt()</i> 2-522 |
| | set NFS UNIX | authentication parameters. | <i>nfsAuthUnixSet()</i> 2-523 |
| | display NFS UNIX | authentication parameters. | <i>nfsAuthUnixShow()</i> 2-523 |
| set | ID number of NFS UNIX | authentication parameters. | <i>nfsIdSet()</i> 2-530 |
| | library. PPP | authentication secrets | <i>pppSecretLib</i> 1-295 |
| | add secret to PPP | authentication secrets table. | <i>pppSecretAdd()</i> 2-587 |
| | delete secret from PPP | authentication secrets table. | <i>pppSecretDelete()</i> 2-588 |
| | display PPP | authentication secrets table. | <i>pppSecretShow()</i> 2-589 |
| | connect routine to | auxiliary clock interrupt. | <i>sysAuxClkConnect()</i> 2-841 |

| | Keyword | Name | Page |
|--|--|------------------------------|-------|
| | turn off auxiliary clock interrupts. | <i>sysAuxClkDisable()</i> | 2-841 |
| | turn on auxiliary clock interrupts. | <i>sysAuxClkEnable()</i> | 2-842 |
| | get auxiliary clock rate. | <i>sysAuxClkRateGet()</i> | 2-842 |
| | set auxiliary clock rate. | <i>sysAuxClkRateSet()</i> | 2-843 |
| | return contents of DUART auxiliary control register. | <i>m68681Acr()</i> | 2-390 |
| | set and clear bits in DUART auxiliary control register. | <i>m68681AcrSetClr()</i> | 2-390 |
| | field. extract backplane address from device | <i>bootBpAnchorExtract()</i> | 2-36 |
| | driver. shared memory backplane network interface | <i>if_sm</i> | 1-166 |
| | change backspace character. | <i>tyBackspaceSet()</i> | 2-928 |
| | compute base-2 logarithm. | <i>log2()</i> | 2-334 |
| | compute base-2 logarithm. | <i>log2f()</i> | 2-335 |
| | set baud rate for SLIP interface. | <i>slipBaudSet()</i> | 2-756 |
| | (ANSI). create temporary binary file (Unimplemented) | <i>tmpfile()</i> | 2-920 |
| | perform binary search (ANSI). | <i>bsearch()</i> | 2-45 |
| | create and initialize binary semaphore. | <i>semBCreate()</i> | 2-708 |
| | and initialize release 4.x binary semaphore. create | <i>semCreate()</i> | 2-710 |
| | initialize static binary semaphore. | <i>semInit()</i> | 2-714 |
| | create and initialize binary semaphore. | <i>VXWBSem::VXWBSem()</i> | 2-978 |
| | binary semaphore library. | <i>semBLib</i> | 1-336 |
| | release 4.x binary semaphore library. | <i>semOLib</i> | 1-344 |
| | /and initialize shared memory binary semaphore (VxMP Opt.). | <i>semBSmCreate()</i> | 2-708 |
| | specified physical/ show BLK_DEV structures on | <i>scsiBlkDevShow()</i> | 2-674 |
| | initialize file system on block device. | <i>diskInit()</i> | 2-137 |
| | logical partition on SCSI block device. define | <i>scsiBlkDevCreate()</i> | 2-673 |
| | read sector(s) from SCSI block device. | <i>scsiRdSecs()</i> | 2-688 |
| | write sector(s) to SCSI block device. | <i>scsiWrtSecs()</i> | 2-702 |
| | library. raw block device file system | <i>rawFsLib</i> | 1-299 |
| | system functions. associate block device with dosFs file | <i>dosFsDevInit()</i> | 2-141 |
| | functions. associate block device with raw volume | <i>rawFsDevInit()</i> | 2-610 |
| | change boot line. | <i>bootChange()</i> | 2-36 |
| | interpret boot parameters from boot line. | <i>bootStringToStruct()</i> | 2-44 |
| | construct boot line. | <i>bootStructToString()</i> | 2-44 |
| | prompt for boot line parameters. | <i>bootParamsPrompt()</i> | 2-39 |
| | display boot line parameters. | <i>bootParamsShow()</i> | 2-39 |
| | line. interpret boot parameters from boot | <i>bootStringToStruct()</i> | 2-44 |
| | retrieve boot parameters using BOOTP. | <i>bootpParamsGet()</i> | 2-41 |
| | boot ROM subroutine library. | <i>bootLib</i> | 1-30 |
| | configuration module for boot ROMs. system | <i>bootConfig</i> | 1-29 |
| | and transfer control to boot ROMs. /network devices | <i>reboot()</i> | 2-616 |
| | network with DHCP at boot time. initialize | <i>dhcpcBootBind()</i> | 2-116 |
| | retrieve boot parameters using BOOTP. | <i>bootpParamsGet()</i> | 2-41 |
| | BOOTP client library. | <i>bootpLib</i> | 1-32 |
| | send BOOTP request message. | <i>bootpMsgSend()</i> | 2-40 |
| | delete breakpoint. | <i>bd()</i> | 2-31 |
| | set hardware breakpoint. | <i>bh()</i> | 2-34 |
| | continue from breakpoint. | <i>c()</i> | 2-47 |
| | breakpoint type (MIPS/ bind breakpoint handler to | <i>dbgBpTypeBind()</i> | 2-108 |
| | bind breakpoint handler to breakpoint type (MIPS R3000,/ | <i>dbgBpTypeBind()</i> | 2-108 |
| | set or display breakpoints. | <i>b()</i> | 2-27 |
| | delete all breakpoints. | <i>bdall()</i> | 2-32 |

| | Keyword | Name | Page |
|--------------------------------|----------------------------|---|--------|
| | interface. get | broadcast address for network <i>ifBroadcastGet()</i> | 2-259 |
| | interface. set | broadcast address for network <i>ifBroadcastSet()</i> | 2-260 |
| | particular port. disable | broadcast forwarding for <i>proxyPortFwdOff()</i> | 2-598 |
| | particular port. enable | broadcast forwarding for <i>proxyPortFwdOn()</i> | 2-598 |
| | change SNTP server | broadcast settings. <i>sntpConfigSet()</i> | 2-797 |
| convert | calendar time into UTC | broken-down time (ANSI). <i>gmtime()</i> | 2-245 |
| | convert calendar time into | broken-down time (ANSI). <i>localtime()</i> | 2-333 |
| | time (ANSI). convert | broken-down time into calendar <i>mktime()</i> | 2-432 |
| | formatted string/ convert | broken-down time into <i>strftime()</i> | 2-822 |
| | (ANSI). convert | broken-down time into string <i>asctime()</i> | 2-18 |
| | (POSIX). convert | broken-down time into string <i>asctime_r()</i> | 2-18 |
| | convert calendar time into | broken-down time (POSIX). <i>gmtime_r()</i> | 2-245 |
| | convert calendar time into | broken-down time (POSIX). <i>localtime_r()</i> | 2-333 |
| | connect | BSP serial device interrupts. <i>sysSerialHwInit2()</i> | 2-858 |
| | state. initialize | BSP serial devices to quiescent <i>sysSerialHwInit()</i> | 2-858 |
| | number. return | BSP version and revision <i>sysBspRev()</i> | 2-843 |
| number of free bytes in ring | | buffer. determine <i>VXWRingBuf::freeBytes()</i> | 2-1010 |
| get characters from ring | | buffer. <i>VXWRingBuf::get()</i> | 2-1010 |
| number of bytes in ring | | buffer. determine <i>VXWRingBuf::nBytes()</i> | 2-1012 |
| put bytes into ring | | buffer. <i>VXWRingBuf::put()</i> | 2-1012 |
| create empty ring | | buffer. <i>VXWRingBuf::VXWRingBuf()</i> | 2-1013 |
| delete ring | | buffer. <i>VXWRingBuf::~~VXWRingBuf()</i> | 2-1014 |
| copy data from zbuf to | | buffer. <i>zbufExtractCopy()</i> | 2-1072 |
| to client and store in | | buffer. /option provided <i>dhcpcOptionGet()</i> | 2-124 |
| read | | buffer. <i>fioRead()</i> | 2-204 |
| to dot notation, store it in | | buffer. /network address <i>inet_ntoa_b()</i> | 2-273 |
| invert order of bytes in | | buffer. <i>binvert()</i> | 2-35 |
| zero out | | buffer. <i>bzero()</i> | 2-46 |
| copy data from mBlk to | | buffer. <i>netMblkToBufCopy()</i> | 2-514 |
| get characters from ring | | buffer. <i>rngBufGet()</i> | 2-646 |
| put bytes into ring | | buffer. <i>rngBufPut()</i> | 2-647 |
| create empty ring | | buffer. <i>rngCreate()</i> | 2-647 |
| delete ring | | buffer. <i>rngDelete()</i> | 2-648 |
| number of free bytes in ring | | buffer. determine <i>rngFreeBytes()</i> | 2-649 |
| number of bytes in ring | | buffer. determine <i>rngNBytes()</i> | 2-651 |
| cacheDmaMalloc(). free | | buffer acquired with <i>cacheDmaFree()</i> | 2-54 |
| interrupt. clean up store | | buffer after data store error <i>cleanUpStoreBuffer()</i> | 2-87 |
| create zbuf segment from | | buffer and insert into zbuf. <i>zbufInsertBuf()</i> | 2-1073 |
| write formatted string to | | buffer (ANSI). <i>sprintf()</i> | 2-800 |
| with variable argument list to | | buffer (ANSI). /formatted <i>vsprintf()</i> | 2-972 |
| ring | | buffer class. <i>VXWRingBuf</i> | 1-422 |
| copy | | buffer data into zbuf. <i>zbufInsertCopy()</i> | 2-1074 |
| time (SPARC). zero out | | buffer eight bytes at a <i>bzeroDoubles()</i> | 2-47 |
| make ring | | buffer empty. <i>VXWRingBuf::flush()</i> | 2-1009 |
| make ring | | buffer empty. <i>rngFlush()</i> | 2-648 |
| drivers. allocate cache-safe | | buffer for DMA devices and <i>cacheDmaMalloc()</i> | 2-54 |
| test whether ring | | buffer is empty. <i>VXWRingBuf::isEmpty()</i> | 2-1010 |
| test if ring | | buffer is empty. <i>rngIsEmpty()</i> | 2-649 |
| test whether ring | | buffer is full (no more room). <i>VXWRingBuf::isFull()</i> | 2-1011 |
| test if ring | | buffer is full (no more room). <i>rngIsFull()</i> | 2-650 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|---------------------------------|--------|
| | network buffer library. | netBufLib | 1-262 |
| | buffer manipulation library. | bLib | 1-28 |
| SPARC assembly language/ | buffer manipulation library. | bALib | 1-27 |
| disable store | buffer (MC68060 only). | cacheStoreBufDisable() | 2-71 |
| enable store | buffer (MC68060 only). | cacheStoreBufEnable() | 2-71 |
| CL_POOL_ID for specified | buffer size. return. | netCIPoolIdGet() | 2-506 |
| ring | buffer subroutine library. | rngLib | 1-308 |
| compare one | buffer to another. | bcmp() | 2-28 |
| copy one | buffer to another. | bcopy() | 2-29 |
| at a time/ copy one | buffer to another eight bytes. | bcopyDoubles() | 2-30 |
| a time. copy one | buffer to another one byte at. | bcopyBytes() | 2-29 |
| word at a time. copy one | buffer to another one long. | bcopyLongs() | 2-30 |
| a time. copy one | buffer to another one word at. | bcopyWords() | 2-31 |
| send encoded | buffer to subagent. | masterIoWrite() | 2-397 |
| character. fill | buffer with specified. | bfill() | 2-32 |
| character one byte at/ fill | buffer with specified. | bfillBytes() | 2-33 |
| eight-byte pattern/ fill | buffer with specified. | bfillDoubles() | 2-33 |
| put byte ahead in ring | buffer without moving ring/ | VXWRingBuf::putAhead() | 2-1013 |
| put byte ahead in ring | buffer without moving ring/ | rngPutAhead() | 2-651 |
| or standard error. set line | buffering for standard output. | setlinebuf() | 2-731 |
| specify | buffering for stream. | setbuffer() | 2-729 |
| specify | buffering for stream (ANSI). | setbuf() | 2-728 |
| specify | buffering for stream (ANSI). | setvbuf() | 2-737 |
| swap | buffers. | bswap() | 2-46 |
| TLBs (Translation Lookaside | Buffers). flush. | pentiumTlbFlush() | 2-566 |
| necessarily/ swap bytes with | buffers that are not. | uswab() | 2-952 |
| flush processor write | buffers to memory. | cachePipeFlush() | 2-68 |
| pulse reset signal on SCSI | bus. | scsiBusReset() | 2-674 |
| test and set location across | bus. | sysBusTas() | 2-845 |
| Databook TCIC/2 PCMCIA host | bus adaptor chip driver. | tcic | 1-390 |
| Intel 82365SL PCMCIA host | bus adaptor chip library. | pcic | 1-280 |
| Intel 82365SL PCMCIA host | bus adaptor chip show library. | pcicShow | 1-281 |
| Databook TCIC/2 PCMCIA host | bus adaptor chip show library. | tcicShow | 1-391 |
| convert local address to | bus address. | sysLocalToBusAdrs() | 2-850 |
| convert | bus address to local address. | sysBusToLocalAdrs() | 2-845 |
| probe address for | bus error. | vxMemProbe() | 2-973 |
| probe address in ASI space for | bus error (SPARC). | vxMemProbeAsi() | 2-974 |
| acknowledge | bus interrupt. | sysBusIntAck() | 2-844 |
| generate | bus interrupt. | sysBusIntGen() | 2-844 |
| disable | bus interrupt level. | sysIntDisable() | 2-849 |
| enable | bus interrupt level. | sysIntEnable() | 2-849 |
| perform PCI | bus scan. | aic7880GetNumOfBuses() | 2-6 |
| assert RST line on SCSI | bus (Western Digital WD33C93/ | sysScsiBusReset() | 2-855 |
| clear all or some entries from | cache. | cacheClear() | 2-50 |
| clear line from CY7C604 | cache. | cacheCy604ClearLine() | 2-51 |
| clear page from CY7C604 | cache. | cacheCy604ClearPage() | 2-51 |
| clear region from CY7C604 | cache. | cacheCy604ClearRegion() | 2-52 |
| clear segment from CY7C604 | cache. | cacheCy604ClearSegment() | 2-52 |
| disable specified | cache. | cacheDisable() | 2-53 |
| enable specified | cache. | cacheEnable() | 2-57 |

| | Keyword | Name | Page |
|--------------------------------|-----------------------------------|-----------------------------------|------|
| flush all or some of specified | cache. | <i>cacheFlush()</i> | 2-57 |
| all or some of specified | cache. invalidate | <i>cacheInvalidate()</i> | 2-65 |
| lock all or part of specified | cache. | <i>cacheLock()</i> | 2-66 |
| clear line from MB86930 | cache. | <i>cacheMb930ClearLine()</i> | 2-66 |
| return size of R3000 data | cache. | <i>cacheR3kDsize()</i> | 2-69 |
| size of R3000 instruction | cache. return | <i>cacheR3kIsize()</i> | 2-69 |
| specific context from Sun-4 | cache. clear | <i>cacheSun4ClearContext()</i> | 2-72 |
| clear line from Sun-4 | cache. | <i>cacheSun4ClearLine()</i> | 2-72 |
| clear page from Sun-4 | cache. | <i>cacheSun4ClearPage()</i> | 2-73 |
| clear segment from Sun-4 | cache. | <i>cacheSun4ClearSegment()</i> | 2-73 |
| all or part of specified | cache. unlock | <i>cacheUnlock()</i> | 2-76 |
| clear entry from | cache (68K, x86). | <i>cacheArchClearEntry()</i> | 2-48 |
| ensure data | cache coherency (i960). | <i>cacheI960JxDCoherent()</i> | 2-60 |
| flush data | cache for drivers. | <i>cacheDrvFlush()</i> | 2-55 |
| invalidate data | cache for drivers. | <i>cacheDrvInvalidate()</i> | 2-55 |
| lock I960Cx 1KB instruction | cache (i960). load and | <i>cacheI960CxIC1kLoadNLock()</i> | 2-58 |
| disable I960Cx instruction | cache (i960). | <i>cacheI960CxICDisable()</i> | 2-58 |
| enable I960Cx instruction | cache (i960). | <i>cacheI960CxICEnable()</i> | 2-58 |
| invalidate I960Cx instruction | cache (i960). | <i>cacheI960CxICInvalidate()</i> | 2-59 |
| I960Cx 512-byte instruction | cache (i960). load and lock | <i>cacheI960CxICLoadNLock()</i> | 2-59 |
| disable I960Jx data | cache (i960). | <i>cacheI960JxDCDisable()</i> | 2-60 |
| enable I960Jx data | cache (i960). | <i>cacheI960JxDCEnable()</i> | 2-61 |
| flush I960Jx data | cache (i960). | <i>cacheI960JxDCFlush()</i> | 2-61 |
| invalidate I960Jx data | cache (i960). | <i>cacheI960JxDCInvalidate()</i> | 2-61 |
| disable I960Jx instruction | cache (i960). | <i>cacheI960JxICDisable()</i> | 2-62 |
| enable I960Jx instruction | cache (i960). | <i>cacheI960JxICEnable()</i> | 2-62 |
| flush I960Jx instruction | cache (i960). | <i>cacheI960JxICFlush()</i> | 2-63 |
| invalidate I960Jx instruction | cache (i960). | <i>cacheI960JxICInvalidate()</i> | 2-63 |
| and lock I960Jx instruction | cache (i960). load | <i>cacheI960JxICLoadNLock()</i> | 2-63 |
| initialize | cache library. | <i>cacheArchLibInit()</i> | 2-48 |
| initialize Cypress CY7C604 | cache library. | <i>cacheCy604LibInit()</i> | 2-53 |
| initialize Fujitsu MB86930 | cache library. | <i>cacheMb930LibInit()</i> | 2-67 |
| initialize microSPARC | cache library. | <i>cacheMicroSparcLibInit()</i> | 2-68 |
| initialize R3000 | cache library. | <i>cacheR3kLibInit()</i> | 2-69 |
| initialize R33000 | cache library. | <i>cacheR33kLibInit()</i> | 2-70 |
| initialize R4000 | cache library. | <i>cacheR4kLibInit()</i> | 2-70 |
| initialize R333x0 | cache library. | <i>cacheR333x0LibInit()</i> | 2-71 |
| initialize Sun-4 | cache library. | <i>cacheSun4LibInit()</i> | 2-74 |
| initialize TI TMS390 | cache library. | <i>cacheTiTms390LibInit()</i> | 2-75 |
| architecture. initialize | cache library for processor | <i>cacheLibInit()</i> | 2-65 |
| initialize I960Cx | cache library (i960). | <i>cacheI960CxLibInit()</i> | 2-60 |
| initialize I960Jx | cache library (i960). | <i>cacheI960JxLibInit()</i> | 2-64 |
| routines. I960Cx | cache management assembly | <i>cacheI960CxALib</i> | 1-35 |
| routines. I960Jx | cache management assembly | <i>cacheI960JxALib</i> | 1-36 |
| routines. MIPS R3000 | cache management assembly | <i>cacheR3kALib</i> | 1-47 |
| architecture-specific | cache management library. | <i>cacheArchLib</i> | 1-34 |
| Cypress CY7C604/605 SPARC | cache management library. | <i>cacheCy604Lib</i> | 1-35 |
| I960Cx | cache management library. | <i>cacheI960CxLib</i> | 1-36 |
| I960Jx | cache management library. | <i>cacheI960JxLib</i> | 1-37 |
| | cache management library. | <i>cacheLib</i> | 1-37 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|---------------------------------|--------|
| Fujitsu MB86930 (SPARC)lite | cache management library. | cacheMb930Lib | 1-46 |
| microSPARC | cache management library. | cacheMicroSparcLib | 1-46 |
| MIPS R3000 | cache management library. | cacheR3kLib | 1-47 |
| MIPS R33000 | cache management library. | cacheR33kLib | 1-48 |
| MIPS R4000 | cache management library. | cacheR4kLib | 1-48 |
| MIPS R333x0 | cache management library. | cacheR333x0Lib | 1-49 |
| Sun-4 | cache management library. | cacheSun4Lib | 1-49 |
| TI TMS390 SuperSPARC | cache management library. | cacheTiTms390Lib | 1-50 |
| get I960Jx data | cache status (i960). | cacheI960JxDCStatusGet() | 2-62 |
| get I960Jx instruction | cache status (i960). | cacheI960JxICStatusGet() | 2-64 |
| instruction and data | caches. synchronize | cacheTextUpdate() | 2-74 |
| synchronize | caches for data coherency. | scsiCacheSynchronize() | 2-676 |
| SCSI that hardware snooping of | caches is disabled. inform | scsiCacheSnoopDisable() | 2-675 |
| SCSI that hardware snooping of | caches is enabled. inform | scsiCacheSnoopEnable() | 2-675 |
| convert broken-down time into | calendar time (ANSI). | mktime() | 2-432 |
| determine current | calendar time (ANSI). | time() | 2-910 |
| time (ANSI). convert | calendar time into broken-down | localtime() | 2-333 |
| time (POSIX). convert | calendar time into broken-down | gmtime_r() | 2-245 |
| time (POSIX). convert | calendar time into broken-down | localtime_r() | 2-333 |
| broken-down time/ convert | calendar time into UTC | gmtime() | 2-245 |
| compute difference between two | calendar times (ANSI). | difftime() | 2-136 |
| transmit single | character. | winIntTx() | 2-1066 |
| fill buffer with specified | character. | bfill() | 2-32 |
| change abort | character. | tyAbortSet() | 2-927 |
| change backspace | character. | tyBackspaceSet() | 2-928 |
| change line-delete | character. | tyDeleteLineSet() | 2-928 |
| change end-of-file | character. | tyEOFSet() | 2-929 |
| change trap-to-monitor | character. | tyMonitorTrapSet() | 2-931 |
| is printing, non-white-space | character (ANSI). /character | isgraph() | 2-306 |
| is printable, including space | character (ANSI). /character | isprint() | 2-307 |
| character is white-space | character (ANSI). /whether | isspace() | 2-308 |
| search block of memory for | character (ANSI). | memchr() | 2-416 |
| stream (ANSI). push | character back into input | ungetc() | 2-940 |
| /string length up to first | character from given set/ | strcspn() | 2-820 |
| first occurrence in string of | character from given set/ find | strpbrk() | 2-825 |
| stream (ANSI). return next | character from standard input | getchar() | 2-239 |
| return next | character from stream (ANSI). | fgetc() | 2-200 |
| return next | character from stream (ANSI). | getc() | 2-239 |
| find first occurrence of | character in string. | index() | 2-268 |
| find last occurrence of | character in string. | rindex() | 2-632 |
| find first occurrence of | character in string (ANSI). | strchr() | 2-818 |
| find last occurrence of | character in string (ANSI). | strrchr() | 2-826 |
| (ANSI). test whether | character is alphanumeric | isalnum() | 2-304 |
| (ANSI). test whether | character is control character | iscntrl() | 2-305 |
| (ANSI). test whether | character is decimal digit | isdigit() | 2-306 |
| (ANSI). test whether | character is hexadecimal digit | isxdigit() | 2-309 |
| test whether | character is letter (ANSI). | isalpha() | 2-304 |
| (ANSI). test whether | character is lower-case letter | islower() | 2-307 |
| including space/ test whether | character is printable, | isprint() | 2-307 |
| non-white-space/ test whether | character is printing, | isgraph() | 2-306 |

| | Keyword | Name | Page |
|--|--|---------------------------|--------|
| | (ANSI). test whether character is punctuation | <i>ispunct()</i> | 2-308 |
| | (ANSI). test whether character is upper-case letter | <i>isupper()</i> | 2-309 |
| | character/ test whether character is white-space | <i>isspace()</i> | 2-308 |
| | /string length up to first character not in given set/ | <i>strspn()</i> | 2-826 |
| | fill buffer with specified character one byte at a/ | <i>bfillBytes()</i> | 2-33 |
| | character/ convert wide character to multibyte | <i>wctomb()</i> | 2-1049 |
| | stream (ANSI). write character to standard output | <i>putchar()</i> | 2-603 |
| | write character to stream (ANSI). | <i>fputc()</i> | 2-219 |
| | write character to stream (ANSI). | <i>putc()</i> | 2-603 |
| | convert multibyte character to wide character/ | <i>mbtowc()</i> | 2-414 |
| | /wide character to multibyte character (Unimplemented)/ | <i>wctomb()</i> | 2-1049 |
| | calculate length of multibyte character (Unimplemented)/ | <i>mblen()</i> | 2-413 |
| | /multibyte character to wide character (Unimplemented)/ | <i>mbtowc()</i> | 2-414 |
| | (ANSI). read and convert characters from ASCII string | <i>sscanf()</i> | 2-808 |
| | another (ANSI). concatenate characters from one string to | <i>strncat()</i> | 2-824 |
| | another (ANSI). copy characters from one string to | <i>strncpy()</i> | 2-825 |
| | get characters from ring buffer. | <i>VXWRingBuf::get()</i> | 2-1010 |
| | get characters from ring buffer. | <i>rngBufGet()</i> | 2-646 |
| | stream (ANSI). read characters from standard input | <i>gets()</i> | 2-242 |
| | stream/ read and convert characters from standard input | <i>scanf()</i> | 2-666 |
| | read specified number of characters from stream (ANSI). | <i>fgets()</i> | 2-201 |
| | read and convert characters from stream (ANSI). | <i>fscanf()</i> | 2-222 |
| | (ANSI). transform up to n characters of s2 into s1 | <i>strxfrm()</i> | 2-833 |
| | (ANSI). compare first n characters of two strings | <i>strcmp()</i> | 2-824 |
| | get information from PC card's CIS. | <i>cisGet()</i> | 2-86 |
| | show CIS information. | <i>cisShow()</i> | 2-87 |
| | PCMCIA CIS library. | <i>cisLib</i> | 1-56 |
| | PCMCIA CIS show library. | <i>cisShow</i> | 1-57 |
| | CL-CD2400 MPCC serial driver. | <i>cd2400Sio</i> | 1-52 |
| | routine to access reference clock. assign | <i>sntpsClockSet()</i> | 2-796 |
| | allocate timer using specified clock for timing base (POSIX). | <i>timer_create()</i> | 2-912 |
| | connect routine to auxiliary clock interrupt. | <i>sysAuxClkConnect()</i> | 2-841 |
| | connect routine to system clock interrupt. | <i>sysClkConnect()</i> | 2-846 |
| | user-defined system clock interrupt routine. | <i>usrClock()</i> | 2-948 |
| | turn off auxiliary clock interrupts. | <i>sysAuxClkDisable()</i> | 2-841 |
| | turn on auxiliary clock interrupts. | <i>sysAuxClkEnable()</i> | 2-842 |
| | turn off system clock interrupts. | <i>sysClkDisable()</i> | 2-846 |
| | turn on system clock interrupts. | <i>sysClkEnable()</i> | 2-847 |
| | clock library (POSIX). | <i>clockLib</i> | 1-57 |
| | get current time of clock (POSIX). | <i>clock_gettime()</i> | 2-89 |
| | get auxiliary clock rate. | <i>sysAuxClkRateGet()</i> | 2-842 |
| | set auxiliary clock rate. | <i>sysAuxClkRateSet()</i> | 2-843 |
| | get system clock rate. | <i>sysClkRateGet()</i> | 2-847 |
| | set system clock rate. | <i>sysClkRateSet()</i> | 2-848 |
| | set clock resolution. | <i>clock_setres()</i> | 2-90 |
| | get clock resolution (POSIX). | <i>clock_getres()</i> | 2-89 |
| | clock tick support library. | <i>tickLib</i> | 1-395 |
| | announce clock tick to kernel. | <i>tickAnnounce()</i> | 2-909 |
| | timer was/ return number of clock ticks elapsed since | <i>envoy_now()</i> | 2-170 |
| | (POSIX). set clock to specified time | <i>clock_settime()</i> | 2-90 |

| | Keyword | Name | Page |
|---|--------------------------------|--------------------------------|--------|
| | close directory (POSIX). | <i>closedir()</i> | 2-91 |
| | close file. | <i>close()</i> | 2-91 |
| | close message queue (POSIX). | <i>mq_close()</i> | 2-448 |
| | close named semaphore (POSIX). | <i>sem_close()</i> | 2-719 |
| | close stream (ANSI). | <i>fclose()</i> | 2-192 |
| buffer size. return | CL_POOL_ID for specified | <i>netCIPoolIdGet()</i> | 2-506 |
| two strings (ANSI). | compare first n characters of | <i>strncmp()</i> | 2-824 |
| | compare one buffer to another. | <i>bcmp()</i> | 2-28 |
| (ANSI). | compare two blocks of memory | <i>memcmp()</i> | 2-416 |
| appropriate to LC_COLLATE/ lexicographically (ANSI). | compare two strings as | <i>strcoll()</i> | 2-819 |
| one string to another (ANSI). | compare two strings | <i>strcmp()</i> | 2-819 |
| another (ANSI). | concatenate characters from | <i>strncat()</i> | 2-824 |
| | concatenate one string to | <i>strcat()</i> | 2-818 |
| | concatenate two lists. | <i>lstConcat()</i> | 2-350 |
| | concatenate two lists. | <i>VXWList::concat()</i> | 2-981 |
| system SCSI | configuration. | <i>sysScsiConfig()</i> | 2-855 |
| changes. alter RIP | configuration after interface | <i>ripIfReset()</i> | 2-638 |
| display dosFs volume | configuration data. | <i>dosFsConfigShow()</i> | 2-140 |
| show volume | configuration information. | <i>cdromFsVolConfigShow()</i> | 2-82 |
| requested NFS device. read | configuration information from | <i>nfsDevInfoGet()</i> | 2-524 |
| user-defined system | configuration library. | <i>usrConfig</i> | 1-410 |
| ROMs. system | configuration module for boot | <i>bootConfig</i> | 1-29 |
| add routine to handle | configuration parameters. | <i>dhcpcEventHookAdd()</i> | 2-121 |
| retrieve current | configuration parameters. | <i>dhcpcParamsGet()</i> | 2-127 |
| handler. remove | configuration parameters | <i>dhcpcEventHookDelete()</i> | 2-122 |
| DHCP. obtain set of network | configuration parameters with | <i>dhcpcBind()</i> | 2-115 |
| run-time client/ Dynamic Host | Configuration Protocol (DHCP) | <i>dhcpcLib</i> | 1-72 |
| server library. Dynamic Host | Configuration Protocol (DHCP) | <i>dhcpsLib</i> | 1-75 |
| /state of DUART output port | configuration register. | <i>m68681Opr()</i> | 2-393 |
| bits in DUART output port | configuration register. /clear | <i>m68681OprSetClr()</i> | 2-394 |
| get PCMCIA | configuration register. | <i>cisConfigregGet()</i> | 2-85 |
| set PCMCIA | configuration register. | <i>cisConfigregSet()</i> | 2-85 |
| initialize dosFs volume | configuration structure. | <i>dosFsConfigInit()</i> | 2-139 |
| obtain dosFs volume | configuration values. | <i>dosFsConfigGet()</i> | 2-138 |
| connected to SCSI controller. | configure all devices | <i>scsiAutoConfig()</i> | 2-672 |
| | configure SCSI peripherals. | <i>usrScsiConfig()</i> | 2-951 |
| | continue from breakpoint. | <i>c()</i> | 2-47 |
| parameters. initiate or | continue negotiating transfer | <i>scsiSyncXferNegotiate()</i> | 2-697 |
| parameters. initiate or | continue negotiating wide | <i>scsiWideXferNegotiate()</i> | 2-701 |
| subroutine returns. | continue until current | <i>cret()</i> | 2-104 |
| test whether character is | control character (ANSI). | <i>iscntrl()</i> | 2-305 |
| for presence of floating-point | coprocessor. probe | <i>fppProbe()</i> | 2-210 |
| restore floating-point | coprocessor context. | <i>fppRestore()</i> | 2-211 |
| save floating-point | coprocessor context. | <i>fppSave()</i> | 2-212 |
| /floating-point | coprocessor support. | <i>fppArchLib</i> | 1-118 |
| initialize floating-point | coprocessor support. | <i>fppInit()</i> | 2-210 |
| floating-point | coprocessor support library. | <i>fppLib</i> | 1-119 |
| | copy buffer data into zbuf. | <i>zbufInsertCopy()</i> | 2-1074 |
| string to another (ANSI). | copy characters from one | <i>strncpy()</i> | 2-825 |
| | copy data from mBlk to buffer. | <i>netMblkToBufCopy()</i> | 2-514 |

| Keyword | Name | Page |
|---|--------------------------------|--------------------------------------|
| copy data from zbuf to buffer. | <i>zbufExtractCopy()</i> | 2-1072 |
| streams. | copy from/to specified | <i>copyStreams()</i> 2-94 |
| stdout. | copy in (or stdin) to out (or | <i>copy()</i> 2-93 |
| to another (ANSI). | copy memory from one location | <i>memcpy()</i> 2-417 |
| to another (ANSI). | copy memory from one location | <i>memmove()</i> 2-421 |
| initialize list as | copy of another. | <i>VXWList::VXWList()</i> 2-987 |
| | copy one buffer to another. | <i>bcopy()</i> 2-29 |
| eight bytes at a time/ byte at a time. | copy one buffer to another | <i>bcopyDoubles()</i> 2-30 |
| long word at a time. | copy one buffer to another one | <i>bcopyBytes()</i> 2-29 |
| word at a time. | copy one buffer to another one | <i>bcopyLongs()</i> 2-30 |
| (ANSI). | copy one buffer to another one | <i>bcopyWords()</i> 2-31 |
| | copy one string to another | <i>strcpy()</i> 2-820 |
| | copy packet to interface.. | <i>elcPut()</i> 2-160 |
| | copy packet to interface.. | <i>enePut()</i> 2-168 |
| | copy packet to interface.. | <i>esmcPut()</i> 2-174 |
| | copy packet to interface.. | <i>ultraPut()</i> 2-939 |
| compute both sine and compute both sine and | cosine. | <i>sincos()</i> 2-753 |
| compute arc | cosine (ANSI). | <i>sincosf()</i> 2-754 |
| compute arc | cosine (ANSI). | <i>acos()</i> 2-3 |
| compute | cosine (ANSI). | <i>acosf()</i> 2-3 |
| compute | cosine (ANSI). | <i>cos()</i> 2-94 |
| compute | cosine (ANSI). | <i>cosf()</i> 2-95 |
| compute hyperbolic | cosine (ANSI). | <i>cosh()</i> 2-95 |
| compute hyperbolic | cosine (ANSI). | <i>coshf()</i> 2-96 |
| half of 64Bit TSC (Timestamp Counter). get lower | | <i>pentiumTscGet32()</i> 2-566 |
| get 64Bit TSC (Timestamp Counter). | | <i>pentiumTscGet64()</i> 2-567 |
| reset TSC (Timestamp Counter). | | <i>pentiumTscReset()</i> 2-567 |
| get value of kernel's tick counter. | | <i>tickGet()</i> 2-909 |
| set value of kernel's tick counter. | | <i>tickSet()</i> 2-910 |
| create and initialize create and initialize | counting semaphore. | <i>semCCreate()</i> 2-709 |
| | counting semaphore. | <i>VXWCSem::VXWCSem()</i> 2-980 |
| | counting semaphore library. | <i>semCLib</i> 1-338 |
| /and initialize shared memory | counting semaphore (VxMP/ | <i>semCSmCreate()</i> 2-710 |
| cancel currently cancel currently | counting watchdog. | <i>VXWWd::cancel()</i> 2-1046 |
| ANSI | counting watchdog. | <i>wdCancel()</i> 2-1061 |
| ctype documentation. | | <i>ansiCtype</i> 1-12 |
| compute | cube root. | <i>cbrt()</i> 2-77 |
| compute | cube root. | <i>cbrtf()</i> 2-78 |
| clear line from clear page from | CY7C604 cache. | <i>cacheCy604ClearLine()</i> 2-51 |
| clear region from clear segment from | CY7C604 cache. | <i>cacheCy604ClearPage()</i> 2-51 |
| initialize Cypress initialize | CY7C604 cache. | <i>cacheCy604ClearRegion()</i> 2-52 |
| cache management library. initialize mount | CY7C604 cache library. | <i>cacheCy604ClearSegment()</i> 2-52 |
| VxWorks remote login VxWorks telnet | CY7C604 cache library. | <i>cacheCy604LibInit()</i> 2-53 |
| initialize telnet TFTP server | Cypress CY7C604/605 SPARC | <i>cacheCy604LibInit()</i> 2-53 |
| | daemon. | <i>cacheCy604Lib</i> 1-35 |
| | daemon. | <i>mountdInit()</i> 2-446 |
| | daemon. | <i>rlogind()</i> 2-644 |
| | daemon. | <i>telnetd()</i> 2-898 |
| | daemon. | <i>telnetInit()</i> 2-899 |
| | daemon task. | <i>tftpdTask()</i> 2-901 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|------------------------------|--------|
| set dosFs file system | date. | <i>dosFsDateSet()</i> | 2-140 |
| set rt11Fs file system | date. | <i>rt11FsDateSet()</i> | 2-658 |
| driver. Set | debug flag in UNIX's ULIP | <i>ulipDebugSet()</i> | 2-934 |
| partition. set | debug options for memory | <i>memPartOptionsSet()</i> | 2-425 |
| memory system partition/ set | debug options for memory | <i>VXWMemPart::options()</i> | 2-990 |
| memory partition. set | debug options for shared | <i>smMemOptionsSet()</i> | 2-762 |
| | debug options for system | <i>memOptionsSet()</i> | 2-421 |
| | debugging facilities. | <i>dbgLib</i> | 1-60 |
| display | debugging help menu. | <i>dbgHelp()</i> | 2-109 |
| protocol. display | debugging information for TCP | <i>tcpDebugShow()</i> | 2-896 |
| specify amount of | debugging output. | <i>ripDebugLevelSet()</i> | 2-636 |
| initialize local | debugging package. | <i>dbgInit()</i> | 2-110 |
| registers 0 thru 15. display | dec 21040/21140 status | <i>dcCsrShow()</i> | 2-111 |
| interface driver. END-style | DEC 21x40 PCI Ethernet network | <i>dec21x40End</i> | 1-67 |
| interface driver. | DEC 21x4x Ethernet LAN network | <i>if_dc</i> | 1-133 |
| interface driver. END style | DEC 21x4x PCI Ethernet network | <i>dec21x4xEnd</i> | 1-63 |
| | delay task from executing. | <i>taskDelay()</i> | 2-866 |
| initialize list | descriptor. | <i>lstInit()</i> | 2-354 |
| initialize rt11Fs device | descriptor. | <i>rt11FsDevInit()</i> | 2-659 |
| initialize tty device | descriptor. | <i>tyDevInit()</i> | 2-929 |
| /shared memory objects | descriptor (VxMP Opt.). | <i>smObjInit()</i> | 2-774 |
| packet to network interface | device. output | <i>cpmStartOutput()</i> | 2-102 |
| initialize pipe packet | device.. | <i>wdbPipePktDevInit()</i> | 2-1056 |
| initialize driver and | device. | <i>dec21x40EndLoad()</i> | 2-113 |
| initialize driver and | device. | <i>dec21x4xEndLoad()</i> | 2-113 |
| and initialize driver and | device. /eex network interface | <i>eexattach()</i> | 2-151 |
| initialize driver and | device. | <i>ei82596EndLoad()</i> | 2-153 |
| and initialize driver and | device. /ei network interface | <i>eiattach()</i> | 2-154 |
| and initialize driver and | device. /ei network interface | <i>eihkattach()</i> | 2-155 |
| initialize driver and | device. | <i>el3c90xEndLoad()</i> | 2-158 |
| and initialize driver and | device. /elc network interface | <i>elcattach()</i> | 2-160 |
| initialize driver and | device. | <i>elt3c509Load()</i> | 2-161 |
| and initialize driver and | device. publish elt interface | <i>eltattach()</i> | 2-163 |
| and initialize driver and | device. /ene network interface | <i>eneattach()</i> | 2-167 |
| initialize driver and | device. | <i>fei82557EndLoad()</i> | 2-196 |
| and initialize driver and | device. /fn network interface | <i>fnattach()</i> | 2-207 |
| initialize driver and | device. | <i>iOlicomEndLoad()</i> | 2-291 |
| underlying driver is tty | device. return whether | <i>isatty()</i> | 2-305 |
| initialize driver and | device. | <i>ln97xEndLoad()</i> | 2-321 |
| initialize driver and | device. | <i>ln7990EndLoad()</i> | 2-323 |
| and initialize driver and | device. /network interface | <i>lnPciattach()</i> | 2-326 |
| initialize driver and | device. | <i>mb86960EndLoad()</i> | 2-403 |
| initialize driver and | device. | <i>mbcEndLoad()</i> | 2-409 |
| packet to network interface | device. output | <i>mbcStartOutput()</i> | 2-412 |
| create memory | device. | <i>memDevCreate()</i> | 2-417 |
| delete memory | device. | <i>memDevDelete()</i> | 2-419 |
| initialize driver and | device. | <i>motCpmEndLoad()</i> | 2-444 |
| initialize driver and | device. | <i>motFecEndLoad()</i> | 2-445 |
| information to MUX or to | device. send control | <i>muxIoctl()</i> | 2-476 |
| address to multicast table for | device. add multicast | <i>muxMCastAddrAdd()</i> | 2-477 |

| | Keyword | Name | Page |
|--------------------------------|--|---------------------------------|-------|
| initialize driver and | device. | <i>ne2000EndLoad()</i> | 2-502 |
| create remote file | device. | <i>netDevCreate()</i> | 2-507 |
| information from requested NFS | device. read configuration | <i>nfsDevInfoGet()</i> | 2-524 |
| unmount NFS | device. | <i>nfsUnmount()</i> | 2-532 |
| initialize driver and | device. | <i>nicEndLoad()</i> | 2-532 |
| initialize driver and | device. | <i>ntLoad()</i> | 2-538 |
| enable PCMCIA-ATA | device. | <i>pccardAtaEnabler()</i> | 2-551 |
| create pipe | device. | <i>pipeDevCreate()</i> | 2-571 |
| create RAM disk | device. | <i>ramDevCreate()</i> | 2-608 |
| read bytes from file or | device. | <i>read()</i> | 2-614 |
| on specified physical | device. /BLK_DEV structures | <i>scsiBlkDevShow()</i> | 2-674 |
| issue ERASE command to SCSI | device. | <i>scsiErase()</i> | 2-677 |
| FORMAT_UNIT command to SCSI | device. issue | <i>scsiFormatUnit()</i> | 2-677 |
| issue INQUIRY command to SCSI | device. | <i>scsiInquiry()</i> | 2-679 |
| command to SCSI | device. issue LOAD/UNLOAD | <i>scsiLoadUnit()</i> | 2-680 |
| MODE_SELECT command to SCSI | device. issue | <i>scsiModeSelect()</i> | 2-684 |
| MODE_SENSE command to SCSI | device. issue | <i>scsiModeSense()</i> | 2-684 |
| information for physical | device. show status | <i>scsiPhysDevShow()</i> | 2-688 |
| bytes or blocks from SCSI | device. read | <i>scsiRdTape()</i> | 2-689 |
| tape | device. issue READ_CAPACITY | <i>scsiReadCapacity()</i> | 2-689 |
| command to SCSI | device. | <i>scsiRelease()</i> | 2-690 |
| issue RELEASE command to SCSI | device. issue | <i>scsiReleaseUnit()</i> | 2-690 |
| issue RESERVE command to SCSI | device. | <i>scsiReserve()</i> | 2-691 |
| RESERVE UNIT command to SCSI | device. issue | <i>scsiReserveUnit()</i> | 2-692 |
| issue REWIND command to SCSI | device. | <i>scsiRewind()</i> | 2-692 |
| create SCSI sequential | device. | <i>scsiSeqDevCreate()</i> | 2-693 |
| command to SCSI | device. /READ_BLOCK_LIMITS | <i>scsiSeqReadBlockLimits()</i> | 2-694 |
| on specified physical SCSI | device. move tape | <i>scsiSpace()</i> | 2-696 |
| command to SCSI | device. issue START_STOP_UNIT | <i>scsiStartStopUnit()</i> | 2-696 |
| command to SCSI tape | device. issue MODE_SELECT | <i>scsiTapeModeSelect()</i> | 2-697 |
| command to SCSI tape | device. issue MODE_SENSE | <i>scsiTapeModeSense()</i> | 2-698 |
| command to SCSI | device. issue TEST_UNIT_READY | <i>scsiTestUnitRdy()</i> | 2-700 |
| file marks to SCSI sequential | device. write | <i>scsiWrtFileMarks()</i> | 2-701 |
| write data to SCSI tape | device. | <i>scsiWrtTape()</i> | 2-702 |
| and initialize driver and | device. /sl network interface | <i>slattach()</i> | 2-756 |
| and initialize driver and | device. publish sm interface | <i>smIfAttach()</i> | 2-758 |
| initialize driver and | device. | <i>sn83932EndLoad()</i> | 2-779 |
| and initialize driver and | device. /sn network interface | <i>snattach()</i> | 2-779 |
| create PCMCIA memory disk | device. | <i>sramDevCreate()</i> | 2-806 |
| create cdromFsLib | device. | <i>cdromFsDevCreate()</i> | 2-81 |
| do task-level read for tty | device. | <i>tyRead()</i> | 2-932 |
| do task-level write for tty | device. | <i>tyWrite()</i> | 2-933 |
| ultra interface and initialize | device. publish | <i>ultraattach()</i> | 2-937 |
| initialize driver and | device. | <i>ultraLoad()</i> | 2-937 |
| create UNIX disk | device. | <i>unixDiskDevCreate()</i> | 2-942 |
| channels. provide terminal | device access to serial | ttyDrv | 1-398 |
| system. initialize | device and create dosFs file | <i>dosFsMkfs()</i> | 2-144 |
| system. initialize | device and create rt11Fs file | <i>rt11FsMkfs()</i> | 2-661 |
| system. initialize | device and mount DOS file | <i>pccardMkfs()</i> | 2-552 |
| command to SCSI | device and read results. issue REQUEST_SENSE . | <i>scsiReqSense()</i> | 2-691 |

| | Keyword | Name | Page |
|--------------------------------|--|------------------------------|--------|
| channel. get | SIO_CHAN device associated with serial | <i>sysSerialChanGet()</i> | 2-857 |
| routine. start | device by calling start | <i>muxDevStart()</i> | 2-474 |
| routine. stop | device by calling stop | <i>muxDevStop()</i> | 2-474 |
| handle | device control requests. | <i>tyIoctl()</i> | 2-930 |
| initialize rt11Fs | device descriptor. | <i>rt11FsDevInit()</i> | 2-659 |
| initialize tty | device descriptor. | <i>tyDevInit()</i> | 2-929 |
| IDE disk | device driver. | <i>ideDrv</i> | 1-125 |
| pseudo memory | device driver. | <i>memDrv</i> | 1-226 |
| (LOCAL and PCMCIA) disk | device driver. ATA/IDE | <i>ataDrv</i> | 1-24 |
| NEC 765 floppy disk | device driver. | <i>nec765Fd</i> | 1-261 |
| PCMCIA SRAM | device driver. | <i>sramDrv</i> | 1-370 |
| poll for packet from | device driver. | <i>muxPollReceive()</i> | 2-481 |
| parallel chip | device driver for IBM-PC LPT. | <i>lptDrv</i> | 1-197 |
| initialize TSFS | device driver for WDB agent. | <i>wdbTsfsDrv()</i> | 2-1058 |
| /(LOCAL and PCMCIA) disk | device driver show routine. | <i>ataShow</i> | 1-27 |
| extract backplane address from | device field. | <i>bootBpAnchorExtract()</i> | 2-36 |
| tape sequential | device file system library. | <i>tapeFsLib</i> | 1-380 |
| create | device for ATA/IDE disk. | <i>ataDevCreate()</i> | 2-20 |
| create | device for floppy disk. | <i>fdDevCreate()</i> | 2-193 |
| create | device for IDE disk. | <i>ideDevCreate()</i> | 2-256 |
| create | device for LPT port. | <i>lptDevCreate()</i> | 2-347 |
| create memory | device for multiple files. | <i>memDevCreateDir()</i> | 2-419 |
| create VxWorks | device for serial channel. | <i>ttyDevCreate()</i> | 2-926 |
| initialize NETROM packet | device for WDB agent. | <i>wdbNetromPktDevInit()</i> | 2-1055 |
| initialize SLIP packet | device for WDB agent. | <i>wdbSlipPktDevInit()</i> | 2-1056 |
| delete | device from I/O system. | <i>iosDevDelete()</i> | 2-293 |
| find I/O | device in device list. | <i>iosDevFind()</i> | 2-294 |
| connect BSP serial | device interrupts. | <i>sysSerialHwInit2()</i> | 2-858 |
| MUX. tests whether | device is already loaded into | <i>muxDevExists()</i> | 2-472 |
| SCSI sequential access | device library (SCSI-2). | <i>scsiSeqLib</i> | 1-334 |
| find I/O device in | device list. | <i>iosDevFind()</i> | 2-294 |
| create SCSI physical | device structure. | <i>scsiPhysDevCreate()</i> | 2-686 |
| add | device to I/O system. | <i>iosDevAdd()</i> | 2-293 |
| find | device using string name. | <i>endFindByName()</i> | 2-166 |
| modify mode of raw | device volume. | <i>rawFsModeChange()</i> | 2-611 |
| disable raw | device volume. | <i>rawFsVolUnmount()</i> | 2-613 |
| disable tape | device volume. | <i>tapeFsVolUnmount()</i> | 2-864 |
| system functions. associate | device with ntPassFs file | <i>ntPassFsDevInit()</i> | 2-539 |
| functions. associate | device with passFs file system | <i>passFsDevInit()</i> | 2-549 |
| associate sequential | device with tape volume/ | <i>tapeFsDevInit()</i> | 2-862 |
| list all system-known | devices. | <i>devs()</i> | 2-115 |
| display mounted NFS | devices. | <i>nfsDevShow()</i> | 2-525 |
| function for sequential access | devices. perform I/O control | <i>scsiSeqIoctl()</i> | 2-694 |
| cache-safe buffer for DMA | devices and drivers. allocate | <i>cacheDmaMalloc()</i> | 2-54 |
| to boot ROMs. reset network | devices and transfer control | <i>reboot()</i> | 2-616 |
| controller. list physical | devices attached to SCSI | <i>scsiShow()</i> | 2-695 |
| controller. configure all | devices connected to SCSI | <i>scsiAutoConfig()</i> | 2-672 |
| display list of | devices in system. | <i>iosDevShow()</i> | 2-294 |
| create list of all NFS | devices in system. | <i>nfsDevListGet()</i> | 2-524 |
| common commands for all | devices (SCSI-2). /library | <i>scsiCommonLib</i> | 1-330 |

| | Keyword | Name | Page |
|---|--------------------------------------|-------------------------------|-------|
| SCSI library for direct access | devices (SCSI-2). | scsiDirectLib | 1-331 |
| initialize BSP serial | devices to quiescent state. | <i>sysSerialHwInit()</i> | 2-858 |
| reset all SIO | devices to quiet state. | <i>sysSerialReset()</i> | 2-859 |
| list contents of | directory. | <i>ls()</i> | 2-348 |
| list contents of RT-11 | directory. | <i>lsOld()</i> | 2-349 |
| make | directory. | <i>mkdir()</i> | 2-431 |
| print current default | directory. | <i>pwd()</i> | 2-605 |
| remove | directory. | <i>rmdir()</i> | 2-646 |
| change default | directory. | <i>cd()</i> | 2-78 |
| do long listing of | directory contents. | <i>ll()</i> | 2-320 |
| list | directory contents via FTP. | <i>ftpLs()</i> | 2-234 |
| (POSIX). open | directory for searching | <i>opendir()</i> | 2-543 |
| delete | directory from access list. | <i>tftpdDirectoryRemove()</i> | 2-900 |
| (POSIX). directory handling library | dirLib | 1-80 | |
| read one entry from | directory (POSIX). | <i>readdir()</i> | 2-614 |
| reset position to start of | directory (POSIX). | <i>rewinddir()</i> | 2-631 |
| close | directory (POSIX). | <i>closedir()</i> | 2-91 |
| add | directory to access list. | <i>tftpdDirectoryAdd()</i> | 2-900 |
| specified number of/ | disassemble and display | <i>l()</i> | 2-312 |
| format | disk. | <i>diskFormat()</i> | 2-136 |
| create device for floppy | disk. | <i>fdDevCreate()</i> | 2-193 |
| create device for ATA/IDE | disk. | <i>ataDevCreate()</i> | 2-20 |
| create device for IDE | disk. | <i>ideDevCreate()</i> | 2-256 |
| DOS file system from ATA hard | disk. mount | <i>usrAtaConfig()</i> | 2-947 |
| DOS file system from floppy | disk. mount | <i>usrFdConfig()</i> | 2-948 |
| DOS file system from IDE hard | disk. mount | <i>usrIdeConfig()</i> | 2-949 |
| create RAM | disk device. | <i>ramDevCreate()</i> | 2-608 |
| create PCMCIA memory | disk device. | <i>sramDevCreate()</i> | 2-806 |
| create UNIX | disk device. | <i>unixDiskDevCreate()</i> | 2-942 |
| IDE | disk device driver. | ideDrv | 1-125 |
| ATA/IDE (LOCAL and PCMCIA) | disk device driver. | ataDrv | 1-24 |
| NEC 765 floppy | disk device driver. | nec765Fd | 1-261 |
| ATA/IDE (LOCAL and PCMCIA) | disk device driver show/ | ataShow | 1-27 |
| RAM | disk driver. | ramDrv | 1-298 |
| initialize floppy | disk driver. | <i>fdDrv()</i> | 2-194 |
| install UNIX | disk driver. | <i>unixDrv()</i> | 2-943 |
| (optional). prepare RAM | disk driver for use | <i>ramDrv()</i> | 2-609 |
| initialize ATA/IDE | disk driver show routine. | <i>ataShowInit()</i> | 2-25 |
| and VxSim for HP). UNIX-file | disk driver (VxSim for Solaris | unixDrv | 1-405 |
| floppy | disk initialization. | usrFd | 1-410 |
| initialize dosFs | disk on top of UNIX. | <i>unixDiskInit()</i> | 2-943 |
| show ATA/IDE | disk parameters. | <i>ataShow()</i> | 2-24 |
| quotient and remainder of | division (ANSI). compute | <i>ldiv()</i> | 2-315 |
| initialize L64862 I/O MMU | DMA data structures (SPARC). | <i>mmuL64862DmaInit()</i> | 2-433 |
| allocate cache-safe buffer for | DMA devices and drivers. | <i>cacheDmaMalloc()</i> | 2-54 |
| microSparc I/II I/O | DMA library. | ioMmuMicroSparcLib | 1-180 |
| MBus-to-SBus Interface: I/O | DMA library (SPARC). /L64862 | mmuL64862Lib | 1-232 |
| initialize | dosFs disk on top of UNIX. | <i>unixDiskInit()</i> | 2-943 |
| initialize device and create | dosFs file system. | <i>dosFsMkfs()</i> | 2-144 |
| set | dosFs file system date. | <i>dosFsDateSet()</i> | 2-140 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|------------------------------|-------|
| associate block device with | dosFs file system functions. | <i>dosFsDevInit()</i> | 2-141 |
| set | dosFs file system time. | <i>dosFsTimeSet()</i> | 2-146 |
| prepare to use | dosFs library. | <i>dosFsInit()</i> | 2-143 |
| status. notify | dosFs of change in ready | <i>dosFsReadyChange()</i> | 2-146 |
| modify mode of | dosFs volume. | <i>dosFsModeChange()</i> | 2-145 |
| unmount | dosFs volume. | <i>dosFsVolUnmount()</i> | 2-148 |
| data. display | dosFs volume configuration | <i>dosFsConfigShow()</i> | 2-140 |
| structure. initialize | dosFs volume configuration | <i>dosFsConfigInit()</i> | 2-139 |
| values. obtain | dosFs volume configuration | <i>dosFsConfigGet()</i> | 2-138 |
| get current | dosFs volume options. | <i>dosFsVolOptionsGet()</i> | 2-147 |
| set | dosFs volume options. | <i>dosFsVolOptionsSet()</i> | 2-147 |
| return very large | double. | <i>infinity()</i> | 2-274 |
| convert string to | double (ANSI). | <i>atof()</i> | 2-26 |
| initial portion of string to | double (ANSI). convert | <i>strtod()</i> | 2-827 |
| transfers. enable | double speed SCSI data | <i>aic7880EnableFast20()</i> | 2-5 |
| library. | doubly linked list subroutine | <i>lstLib</i> | 1-198 |
| driver. Nat. Semi | DP83932B SONIC Ethernet | <i>sn83932End</i> | 1-364 |
| National Semiconductor | DP83932B SONIC Ethernet/ | <i>if_sn</i> | 1-167 |
| Ethernet network interface | driver. END style Intel 82557 | <i>fei82557End</i> | 1-114 |
| I8250 serial | driver. | <i>i8250Sio</i> | 1-123 |
| IDE disk device | driver. | <i>ideDrv</i> | 1-125 |
| CPM core network interface | driver. Motorola | <i>if_cpm</i> | 1-126 |
| CS8900 network interface | driver. Crystal Semiconductor | <i>if_cs</i> | 1-129 |
| Ethernet LAN network interface | driver. DEC 21x4x | <i>if_dc</i> | 1-133 |
| 16 network interface | driver. Intel EtherExpress | <i>if_eex</i> | 1-136 |
| Ethernet network interface | driver. Intel 82596 | <i>if_ei</i> | 1-137 |
| Ethernet network interface | driver. SMC 8013WC | <i>if_elc</i> | 1-144 |
| Ethernet network interface | driver. 3Com 3C509 | <i>if_elt</i> | 1-144 |
| NE2000 network interface | driver. Novell/Eagle | <i>if_ene</i> | 1-146 |
| Ethernet network interface | driver. /Ethernet2 SMC-91c9x | <i>if_esmc</i> | 1-147 |
| Ethernet network interface | driver. Intel 82557 | <i>if_fei</i> | 1-149 |
| Ethernet network interface | driver. Fujitsu MB86960 NICE | <i>if_fn</i> | 1-151 |
| Ethernet network interface | driver. AMD Am7990 LANCE | <i>if_ln</i> | 1-153 |
| Ethernet network interface | driver. /Am79C970 PCnet-PCI | <i>if_lnPci</i> | 1-156 |
| loopback network interface | driver. software | <i>if_loop</i> | 1-160 |
| 68EN302 network-interface | driver. Motorola | <i>if_mbc</i> | 1-160 |
| ST-NIC Chip network interface | driver. /Semiconductor | <i>if_nicEvb</i> | 1-163 |
| IP (SLIP) network interface | driver. Serial Line | <i>if_sl</i> | 1-164 |
| backplane network interface | driver. shared memory | <i>if_sm</i> | 1-166 |
| SONIC Ethernet network | driver. /DP83932B | <i>if_sn</i> | 1-167 |
| Ethernet network interface | driver. SMC Elite Ultra | <i>if_ultra</i> | 1-171 |
| PCMCIA network interface | driver. /style Intel Olicom | <i>iOlicomEnd</i> | 1-177 |
| Am79C97X PCnet-PCI Ethernet | driver. END style AMD | <i>ln97xEnd</i> | 1-186 |
| Ethernet network interface | driver. /style AMD 7990 LANCE | <i>ln7990End</i> | 1-190 |
| Motorola MC68302 bimodal tty | driver. | <i>m68302Sio</i> | 1-210 |
| Motorola MC68332 tty | driver. | <i>m68332Sio</i> | 1-211 |
| MC68360 SCC UART serial | driver. Motorola | <i>m68360Sio</i> | 1-211 |
| MC68562 DUSCC serial | driver. | <i>m68562Sio</i> | 1-212 |
| M68681 serial communications | driver. | <i>m68681Sio</i> | 1-213 |
| MC68901 MFP tty | driver. | <i>m68901Sio</i> | 1-215 |

| | Keyword | Name | Page |
|--|---------------------------------------|-----------------------------------|-------------------------------|
| | MB86940 UART tty driver | mb86940Sio | 1-221 |
| | Ethernet network interface driver | /Fujitsu MB86960 | mb86960End 1-221 |
| | END network interface driver | Motorola 68302fads | mbcEnd 1-223 |
| | pseudo memory device driver | | memDrv 1-226 |
| | network interface driver | /MC68EN360/MPC800 | motCpmEnd 1-238 |
| | FEC Ethernet network interface driver | END style Motorola | motFecEnd 1-242 |
| | disk device driver | ATA/IDE (LOCAL and PCMCIA) | ataDrv 1-24 |
| | NE2000 END network interface driver | | ne2000End 1-259 |
| | NEC 765 floppy disk device driver | | nec765Fd 1-261 |
| | network remote file I/O driver | | netDrv 1-263 |
| | Network File System (NFS) I/O driver | | nfsDrv 1-268 |
| | ST-NIC Chip network interface driver | /Semiconductor | nicEvbEnd 1-271 |
| | NS 16550 UART tty driver | | ns16550Sio 1-273 |
| | pipe I/O driver | | pipeDrv 1-289 |
| | ppc403GA serial driver | | ppc403Sio 1-291 |
| | MPC800 SMC UART serial driver | Motorola | ppc860Sio 1-292 |
| | pseudo-terminal driver | | ptyDrv 1-297 |
| | RAM disk driver | | ramDrv 1-298 |
| | Semiconductor SA-1100 UART tty driver | Digital | sa1100Sio 1-316 |
| | memory network (backplane) driver | /interface to shared | smNetLib 1-360 |
| | Semi DP83932B SONIC Ethernet driver | Nat. | sn83932End 1-364 |
| | PCMCIA SRAM device driver | | sramDrv 1-370 |
| | ST 16C552 DUART tty driver | | st16552Sio 1-371 |
| | PCMCIA host bus adaptor chip driver | Databook TCIC/2 | tcic 1-390 |
| | Elite END network interface driver | SMC Ultra | ultraEnd 1-404 |
| | unix serial driver | | unixSio 1-407 |
| | interface for ULIP driver | WDB communication | wdbUlipPktDrv 1-439 |
| | win serial driver | | winSio 1-442 |
| | Communications Controller driver | Z8530 SCC Serial | z8530Sio 1-443 |
| | CL-CD2400 MPCC serial driver | | cd2400Sio 1-52 |
| | PCI Ethernet network interface driver | END style DEC 21x4x | dec21x4xEnd 1-63 |
| | PCI Ethernet network interface driver | END-style DEC 21x40 | dec21x40End 1-67 |
| | Ethernet network interface driver | END style Intel 82596 | ei82596End 1-96 |
| | AIO system driver | | aioSysDrv 1-9 |
| | ARM AMBA UART tty driver | | ambaSio 1-9 |
| | interface and initialize driver | publish cpm network | cpmattach() 2-101 |
| | interface and initialize driver | publish cs network | csAttach() 2-105 |
| | interface and initialize driver | publish esmc network | esmcattach() 2-174 |
| | of multicast addresses from driver | retrieve table | etherMultiGet() 2-179 |
| | initialize floppy disk driver | | fdDrv() 2-194 |
| | initialize ATA driver | | ataDrv() 2-21 |
| | initialize IDE driver | | ideDrv() 2-256 |
| | install I/O driver | | iosDrvInstall() 2-295 |
| | remove I/O driver | | iosDrvRemove() 2-295 |
| | initialize LPT driver | | lptDrv() 2-347 |
| | interface and initialize driver | publish mbc network | mbcattach() 2-408 |
| | install memory driver | | memDrv() 2-420 |
| | poll for packet from device driver | | muxPollReceive() 2-481 |
| | detach protocol from specified driver | | muxUnbind() 2-484 |
| | install network remote file driver | | netDrv() 2-507 |

| | Keyword | Name | Page |
|-------------------------------|--------------------------------------|----------------------------|-------|
| install NFS | driver. | <i>nfsDrv()</i> | 2-526 |
| system driver number for nfs | driver. return IO | <i>nfsDrvNumGet()</i> | 2-526 |
| nicEvb network interface | driver. /and initialize | <i>nicEvbattach()</i> | 2-533 |
| enable PCMCIA-SRAM | driver. | <i>pccardSramEnabler()</i> | 2-553 |
| enable PCMCIA-TFFS | driver. | <i>pccardTffsEnabler()</i> | 2-553 |
| initialize pipe | driver. | <i>pipeDrv()</i> | 2-572 |
| initialize pseudo-terminal | driver. | <i>ptyDrv()</i> | 2-602 |
| shared memory network | driver. initialize | <i>smNetInit()</i> | 2-771 |
| install PCMCIA SRAM memory | driver. | <i>sramDrv()</i> | 2-807 |
| initialize tty | driver. | <i>ttyDrv()</i> | 2-926 |
| Set debug flag in UNIX's ULIP | driver. | <i>ulipDebugSet()</i> | 2-934 |
| install UNIX disk | driver. | <i>unixDrv()</i> | 2-943 |
| initialize AIO system | driver. | <i>aioSysInit()</i> | 2-9 |
| initialize | driver and device. | <i>dec21x40EndLoad()</i> | 2-113 |
| initialize | driver and device. | <i>dec21x4xEndLoad()</i> | 2-113 |
| interface and initialize | driver and device. /network | <i>eexattach()</i> | 2-151 |
| initialize | driver and device. | <i>ei82596EndLoad()</i> | 2-153 |
| interface and initialize | driver and device. /ei network | <i>eiattach()</i> | 2-154 |
| interface and initialize | driver and device. /ei network | <i>eihkattach()</i> | 2-155 |
| initialize | driver and device. | <i>el3c90xEndLoad()</i> | 2-158 |
| interface and initialize | driver and device. /network | <i>elcattach()</i> | 2-160 |
| initialize | driver and device. | <i>elt3c509Load()</i> | 2-161 |
| elt interface and initialize | driver and device. publish | <i>eltattach()</i> | 2-163 |
| interface and initialize | driver and device. /network | <i>eneattach()</i> | 2-167 |
| initialize | driver and device. | <i>fei82557EndLoad()</i> | 2-196 |
| interface and initialize | driver and device. /fn network | <i>fnattach()</i> | 2-207 |
| initialize | driver and device. | <i>iOlicomEndLoad()</i> | 2-291 |
| initialize | driver and device. | <i>ln97xEndLoad()</i> | 2-321 |
| initialize | driver and device. | <i>ln7990EndLoad()</i> | 2-323 |
| interface and initialize | driver and device. /network | <i>lnPciattach()</i> | 2-326 |
| initialize | driver and device. | <i>mb86960EndLoad()</i> | 2-403 |
| initialize | driver and device. | <i>mbcEndLoad()</i> | 2-409 |
| initialize | driver and device. | <i>motCpmEndLoad()</i> | 2-444 |
| initialize | driver and device. | <i>motFecEndLoad()</i> | 2-445 |
| initialize | driver and device. | <i>ne2000EndLoad()</i> | 2-502 |
| initialize | driver and device. | <i>nicEndLoad()</i> | 2-532 |
| initialize | driver and device. | <i>ntLoad()</i> | 2-538 |
| interface and initialize | driver and device. /sl network | <i>slattach()</i> | 2-756 |
| sm interface and initialize | driver and device. publish | <i>smIfAttach()</i> | 2-758 |
| initialize | driver and device. | <i>sn83932EndLoad()</i> | 2-779 |
| interface and initialize | driver and device. /sn network | <i>snattach()</i> | 2-779 |
| initialize | driver and device. | <i>ultraLoad()</i> | 2-937 |
| /interface and initialize | driver and pseudo-device. | <i>loattach()</i> | 2-330 |
| END network interface | driver for 3COM 3C509. | <i>elt3c509End</i> | 1-104 |
| END network interface | driver for 3COM 3C90xB XL. | <i>el3c90xEnd</i> | 1-99 |
| Ethernet network interface | driver for hkv3500. /82596 | <i>if_eihk</i> | 1-140 |
| evaluation. NS16550 serial | driver for IBM PPC403GA | <i>evbNs16550Sio</i> | 1-111 |
| parallel chip device | driver for IBM-PC LPT. | <i>lptDrv</i> | 1-197 |
| END based packet | driver for lightweight UDP/IP. | <i>wdbEndPktDrv</i> | 1-432 |
| pipe packet | driver for lightweight UDP/IP. | <i>wdbPipePktDrv</i> | 1-433 |

| | Keyword | Name | Page |
|--------------------------------|---|----------------------------------|--------|
| | prepare RAM disk driver for use (optional). | <i>ramDrv()</i> | 2-609 |
| (VxSim). network interface | driver for User Level IP | <i>if_ulip</i> | 1-170 |
| NETROM packet | driver for WDB agent. | <i>wdbNetromPktDrv</i> | 1-433 |
| virtual generic file I/O | driver for WDB agent. | <i>wdbTsfsDrv</i> | 1-435 |
| virtual tty I/O | driver for WDB agent. | <i>wdbVioDrv</i> | 1-440 |
| initialize TSFS device | driver for WDB agent. | <i>wdbTsfsDrv()</i> | 2-1058 |
| initialize tty | driver for WDB agent. | <i>wdbVioDrv()</i> | 2-1060 |
| remove | driver from MUX. | <i>muxDevUnload()</i> | 2-475 |
| install | driver function table. | <i>mb86940DevInit()</i> | 2-403 |
| load | driver into MUX. | <i>muxDevLoad()</i> | 2-473 |
| return whether underlying | driver is tty device. | <i>isatty()</i> | 2-305 |
| bind protocol to MUX given | driver name. | <i>muxBind()</i> | 2-471 |
| return IO system | driver number for nfs driver. | <i>nfsDrvNumGet()</i> | 2-526 |
| (LOCAL and PCMCIA) disk device | driver show routine. ATA/IDE | <i>ataShow</i> | 1-27 |
| initialize ATA/IDE disk | driver show routine. | <i>ataShowInit()</i> | 2-25 |
| shared memory network | driver show routines. | <i>smNetShow</i> | 1-361 |
| interface and initialize | driver structures. /ln network | <i>lnattach()</i> | 2-325 |
| tty | driver support library. | <i>tyLib</i> | 1-399 |
| Windows/ END network interface | driver to ULIP for vxSim for | <i>ntEnd</i> | 1-274 |
| VxSim for HP). UNIX-file disk | driver (VxSim for Solaris and | <i>unixDrv</i> | 1-405 |
| support library for END-based | drivers. | <i>endLib</i> | 1-106 |
| display list of system | drivers. | <i>iosDrvShow()</i> | 2-296 |
| configured Enhanced Network | Drivers. all | <i>muxShow()</i> | 2-483 |
| buffer for DMA devices and | drivers. allocate cache-safe | <i>cacheDmaMalloc()</i> | 2-54 |
| all show routines for PCMCIA | drivers. initialize | <i>pcmciaShowInit()</i> | 2-556 |
| flush data cache for | drivers. | <i>cacheDrvFlush()</i> | 2-55 |
| invalidate data cache for | drivers. | <i>cacheDrvInvalidate()</i> | 2-55 |
| translate physical address for | drivers. | <i>cacheDrvPhysToVirt()</i> | 2-56 |
| translate virtual address for | drivers. | <i>cacheDrvVirtToPhys()</i> | 2-56 |
| translate physical address for | drivers. | <i>cacheTiTms390PhysToVirt()</i> | 2-75 |
| register. return contents of | DUART auxiliary control | <i>m68681Acr()</i> | 2-390 |
| set and clear bits in | DUART auxiliary control/ | <i>m68681AcrSetClr()</i> | 2-390 |
| return current contents of | DUART interrupt-mask register. | <i>m68681Imr()</i> | 2-392 |
| set and clear bits in | DUART interrupt-mask register. | <i>m68681ImrSetClr()</i> | 2-392 |
| vector. handle all | DUART interrupts in one | <i>m68681Int()</i> | 2-393 |
| configuration/ return state of | DUART output port | <i>m68681Opcr()</i> | 2-393 |
| set and clear bits in | DUART output port/ | <i>m68681OpcrSetClr()</i> | 2-394 |
| return current state of | DUART output port register. | <i>m68681Opr()</i> | 2-394 |
| set and clear bits in | DUART output port register. | <i>m68681OprSetClr()</i> | 2-395 |
| ST 16C552 | DUART tty driver. | <i>st16552Sio</i> | 1-371 |
| initialize | DUSCC. | <i>m68562HrdInit()</i> | 2-388 |
| MC68562 | DUSCC serial driver. | <i>m68562Sio</i> | 1-212 |
| initialize driver and/ publish | eex network interface and | <i>eexattach()</i> | 2-151 |
| initialize driver and/ publish | ei network interface and | <i>eiattach()</i> | 2-154 |
| initialize driver and/ publish | ei network interface and | <i>eihkattach()</i> | 2-155 |
| /statistics for SMC 8013WC | elc network interface. | <i>elcShow()</i> | 2-161 |
| initialize driver and/ publish | elc network interface and | <i>elcattach()</i> | 2-160 |
| driver. SMC Ultra | Elite END network interface | <i>ultraEnd</i> | 1-404 |
| interface driver. SMC | Elite Ultra Ethernet network | <i>if_ultra</i> | 1-171 |
| driver and device. publish | elt interface and initialize | <i>eltattach()</i> | 2-163 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|------------------------------|--------|
| display statistics for 3C509 | elt network interface. | <i>eltShow()</i> | 2-163 |
| default password | encryption routine. | <i>loginDefaultEncrypt()</i> | 2-338 |
| install | encryption routine. | <i>loginEncryptInstall()</i> | 2-339 |
| lightweight UDP/IP. | END based packet driver for | <i>wdbEndPktDrv</i> | 1-432 |
| Motorola 68302fads | END network interface driver. | <i>mbcEnd</i> | 1-223 |
| NE2000 | END network interface driver. | <i>ne2000End</i> | 1-259 |
| SMC Ultra Elite | END network interface driver. | <i>ultraEnd</i> | 1-404 |
| for 3COM 3C509. | END network interface driver | <i>elt3c509End</i> | 1-104 |
| for 3COM 3C90xB XL. | END network interface driver | <i>el3c90xEnd</i> | 1-99 |
| to ULIP for vxSim for Windows/ | END network interface driver | <i>ntEnd</i> | 1-274 |
| add node to | end of list. | <i>lstAdd()</i> | 2-350 |
| add node to | end of list. | <i>VXWList::add()</i> | 2-981 |
| define address for other | end of point-to-point link. | <i>ifDstAddrSet()</i> | 2-261 |
| Ethernet network interface/ | END style AMD 7990 LANCE | <i>In7990End</i> | 1-190 |
| PCnet-PCI Ethernet driver. | END style AMD Am79C97X | <i>In97xEnd</i> | 1-186 |
| Ethernet network interface/ | END style DEC 21x4x PCI | <i>dec21x4xEnd</i> | 1-63 |
| network interface driver. | END style Intel 82557 Ethernet | <i>fei82557End</i> | 1-114 |
| network interface driver. | END style Intel 82596 Ethernet | <i>ei82596End</i> | 1-96 |
| network interface driver. | END style Intel Olicom PCMCIA | <i>iOlicomEnd</i> | 1-177 |
| Ethernet network interface/ | END style Motorola FEC | <i>motFecEnd</i> | 1-242 |
| MIC68EN360/MPC800 network/ | END style Motorola | <i>motCpmEnd</i> | 1-238 |
| set flags member of | END_OBJ structure. | <i>endObjFlagSet()</i> | 2-166 |
| initialize | END_OBJ structure. | <i>endObjInit()</i> | 2-167 |
| for stream (ANSI). clear | end-of-file and error flags | <i>clearerr()</i> | 2-88 |
| change | end-of-file character. | <i>tyEOFSet()</i> | 2-929 |
| stream (ANSI). test | end-of-file indicator for | <i>feof()</i> | 2-198 |
| display statistics for NE2000 | ene network interface. | <i>eneShow()</i> | 2-168 |
| initialize driver and/ publish | ene network interface and | <i>eneattach()</i> | 2-167 |
| issue | ERASE command to SCSI device. | <i>scsiErase()</i> | 2-677 |
| map error number in | errno to error message (ANSI). | <i>perror()</i> | 2-569 |
| standard output or standard | error. set line buffering for | <i>setlinebuf()</i> | 2-731 |
| probe address for bus | error. | <i>vxMemProbe()</i> | 2-973 |
| change MIB-II | error count. | <i>mib2ErrorAdd()</i> | 2-430 |
| clear end-of-file and | error flags for stream (ANSI). | <i>clearerr()</i> | 2-88 |
| /(take) semaphore, returning | error if unavailable (POSIX). | <i>sem_trywait()</i> | 2-723 |
| pointer (ANSI). test | error indicator for file | <i>ferror()</i> | 2-199 |
| handle receiver/transmitter | error interrupt. | <i>m68562RxTxErrInt()</i> | 2-389 |
| store buffer after data store | error interrupt. clean up | <i>cleanUpStoreBuffer()</i> | 2-87 |
| handle | error interrupts. | <i>z8530IntEx()</i> | 2-1068 |
| handle | error interrupts. | <i>ppc403IntEx()</i> | 2-575 |
| log formatted | error message. | <i>logMsg()</i> | 2-344 |
| map error number in errno to | error message (ANSI). | <i>perror()</i> | 2-569 |
| message (ANSI). map | error number in errno to error | <i>perror()</i> | 2-569 |
| (ANSI). map | error number to error string | <i>strerror()</i> | 2-821 |
| (POSIX). map | error number to error string | <i>strerror_r()</i> | 2-821 |
| address in ASI space for bus | error (SPARC). probe | <i>vxMemProbeAsi()</i> | 2-974 |
| | error status library. | <i>errnoLib</i> | 1-107 |
| I/O operation/ retrieve | error status of asynchronous | <i>aio_error()</i> | 2-9 |
| retrieve | error status value. | <i>VXWTask::errNo()</i> | 2-1028 |
| set | error status value. | <i>VXWTask::errNo()</i> | 2-1028 |

| | Keyword | Name | Page |
|-------------------------------|--------------------------------------|------------------------------|--------|
| print definition of specified | error status value. | <i>printErrno()</i> | 2-591 |
| task. get | error status value of calling | <i>errnoGet()</i> | 2-172 |
| task. set | error status value of calling | <i>errnoSet()</i> | 2-173 |
| specified task. get | error status value of | <i>errnoOfTaskGet()</i> | 2-172 |
| specified task. set | error status value of | <i>errnoOfTaskSet()</i> | 2-173 |
| formatted string to standard | error stream. write | <i>printErr()</i> | 2-590 |
| map error number to | error string (ANSI). | <i>strerror()</i> | 2-821 |
| map error number to | error string (POSIX). | <i>strerror_r()</i> | 2-821 |
| interface driver. Intel | EtherExpress 16 network | <i>if_eex</i> | 1-136 |
| Internet address. resolve | Ethernet address for specified | <i>etherAddrResolve()</i> | 2-175 |
| form | Ethernet address into packet. | <i>endEtherAddressForm()</i> | 2-164 |
| style AMD Am79C97X PCnet-PCI | Ethernet driver. END | <i>ln97xEnd</i> | 1-186 |
| Nat. Semi DP83932B SONIC | Ethernet driver. | <i>sn83932End</i> | 1-364 |
| add routine to receive all | Ethernet input packets. | <i>etherInputHookAdd()</i> | 2-176 |
| send packet on | Ethernet interface. | <i>etherOutput()</i> | 2-179 |
| driver. DEC 21x4x | Ethernet LAN network interface | <i>if_dc</i> | 1-133 |
| record. delete | Ethernet multicast address | <i>etherMultiDel()</i> | 2-178 |
| library to handle | Ethernet multicast addresses. | <i>etherMultiLib</i> | 1-111 |
| /Semiconductor DP83932B SONIC | Ethernet network driver. | <i>if_sn</i> | 1-167 |
| driver. END style Intel 82557 | Ethernet network interface | <i>fei82557End</i> | 1-114 |
| driver. Intel 82596 | Ethernet network interface | <i>if_ei</i> | 1-137 |
| driver. SMC 8013WC | Ethernet network interface | <i>if_elc</i> | 1-144 |
| driver. 3Com 3C509 | Ethernet network interface | <i>if_elt</i> | 1-144 |
| Ampro Ethernet2 SMC-91c9x | Ethernet network interface/ | <i>if_esmc</i> | 1-147 |
| driver. Intel 82557 | Ethernet network interface | <i>if_fei</i> | 1-149 |
| driver. Fujitsu MB86960 NICE | Ethernet network interface | <i>if_fn</i> | 1-151 |
| driver. AMD Am7990 LANCE | Ethernet network interface | <i>if_ln</i> | 1-153 |
| AMD Am79C970 PCnet-PCI | Ethernet network interface/ | <i>if_lnci</i> | 1-156 |
| driver. SMC Elite Ultra | Ethernet network interface | <i>if_ultra</i> | 1-171 |
| END style AMD 7990 LANCE | Ethernet network interface/ | <i>ln7990End</i> | 1-190 |
| END-style Fujitsu MB86960 | Ethernet network interface/ | <i>mb86960End</i> | 1-221 |
| END style Motorola FEC | Ethernet network interface/ | <i>motFecEnd</i> | 1-242 |
| END style DEC 21x4x PCI | Ethernet network interface/ | <i>dec21x4xEnd</i> | 1-63 |
| END-style DEC 21x40 PCI | Ethernet network interface/ | <i>dec21x40End</i> | 1-67 |
| driver. END style Intel 82596 | Ethernet network interface | <i>ei82596End</i> | 1-96 |
| driver for/ Intel 82596 | Ethernet network interface | <i>if_eihk</i> | 1-140 |
| add routine to receive all | Ethernet output packets. | <i>etherOutputHookAdd()</i> | 2-180 |
| get type from | ethernet packet. | <i>etherTypeGet()</i> | 2-182 |
| hooks. | Ethernet raw I/O routines and | <i>etherLib</i> | 1-109 |
| handle controller-bus reset | event. | <i>scsiMgrBusReset()</i> | 2-681 |
| manager of SCSI (controller) | event. notify SCSI | <i>scsiMgrEventNotify()</i> | 2-682 |
| WDB user | event library. | <i>wdbUserEvtLib</i> | 1-439 |
| include WDB user | event library. | <i>wdbUserEvtLibInit()</i> | 2-1059 |
| post user | event string to host tools. | <i>wdbUserEvtPost()</i> | 2-1059 |
| machine. send | event to SCSI controller state | <i>scsiMgrCtrlEvent()</i> | 2-681 |
| send | event to thread state machine. | <i>scsiMgrThreadEvent()</i> | 2-683 |
| generic | exception handling facilities. | <i>exclib</i> | 1-112 |
| initialize | exception handling package. | <i>exclibInit()</i> | 2-185 |
| connect C routine to | exception vector (PowerPC). | <i>excConnect()</i> | 2-183 |
| connect C routine to critical | exception vector (PowerPC/ | <i>excCrtConnect()</i> | 2-184 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------|----------------------------------|--------|
| /C routine to asynchronous | exception vector (PowerPC,/ | <i>excIntConnect()</i> | 2-186 |
| ARM). get CPU | exception vector (PowerPC, | <i>excVecGet()</i> | 2-188 |
| ARM). set CPU | exception vector (PowerPC, | <i>excVecSet()</i> | 2-189 |
| (MC680x0,/ write-protect | exception vector table | <i>intVecTableWriteProtect()</i> | 2-288 |
| initialize | exception/interrupt vectors. | <i>excVecInit()</i> | 2-188 |
| routine to be called with | exceptions. specify | <i>excHookAdd()</i> | 2-185 |
| handle task-level | exceptions. | <i>excTask()</i> | 2-188 |
| | exit task (ANSI). | <i>exit()</i> | 2-190 |
| compute | exponential value (ANSI). | <i>exp()</i> | 2-190 |
| compute | exponential value (ANSI). | <i>expf()</i> | 2-191 |
| specify file system to be NFS | exported. | <i>nfsExport()</i> | 2-528 |
| mount all file systems | exported by specified host. | <i>nfsMountAll()</i> | 2-531 |
| file system from list of | exported file systems. remove | <i>nfsUnexport()</i> | 2-531 |
| remote host. display | exported file systems of | <i>nfsExportShow()</i> | 2-528 |
| write formatted string to | fd. | <i>fdprintf()</i> | 2-195 |
| add logging | fd. | <i>logFdAdd()</i> | 2-337 |
| delete logging | fd. | <i>logFdDelete()</i> | 2-337 |
| set primary logging | fd. | <i>logFdSet()</i> | 2-338 |
| with variable argument list to | fd. write string formatted | <i>vfprintf()</i> | 2-956 |
| value. validate open | fd and return driver-specific | <i>iosFdValue()</i> | 2-296 |
| input/output/error. get | fd for global standard | <i>ioGlobalStdGet()</i> | 2-290 |
| input/output/error. set | fd for global standard | <i>ioGlobalStdSet()</i> | 2-290 |
| return | fd for stream (POSIX). | <i>fileno()</i> | 2-201 |
| input/output/error. get | fd for task standard | <i>ioTaskStdGet()</i> | 2-298 |
| input/output/error. set | fd for task standard | <i>ioTaskStdSet()</i> | 2-298 |
| display list of | fd names in system. | <i>iosFdShow()</i> | 2-296 |
| open file specified by | fd (POSIX). | <i>fdopen()</i> | 2-195 |
| pend on set of | fds. | <i>select()</i> | 2-703 |
| default input/output/error | fds. set shell's | <i>shellOrigStdSet()</i> | 2-740 |
| driver. END style Motorola | FEC Ethernet network interface | <i>motFecEnd</i> | 1-242 |
| 7880 SCSI Host Adapter Library | File. Adaptec | <i>aic7880Lib</i> | 1-1 |
| create | file. | <i>creat()</i> | 2-104 |
| write bytes to | file. | <i>write()</i> | 2-1066 |
| read string from | file. | <i>fioRdString()</i> | 2-203 |
| open | file. | <i>open()</i> | 2-542 |
| change name of | file. | <i>rename()</i> | 2-621 |
| remove | file. | <i>rm()</i> | 2-645 |
| close | file. | <i>close()</i> | 2-91 |
| update time on | file. | <i>utime()</i> | 2-953 |
| remove | file (ANSI). | <i>remove()</i> | 2-621 |
| indicator to beginning of | file (ANSI). /file position | <i>rewind()</i> | 2-631 |
| create remote | file device. | <i>netDevCreate()</i> | 2-507 |
| install network remote | file driver. | <i>netDrv()</i> | 2-507 |
| get | file from remote system. | <i>tftpGet()</i> | 2-902 |
| network remote | file I/O driver. | <i>netDrv</i> | 1-263 |
| virtual generic | file I/O driver for WDB agent. | <i>wdbTsfsDrv</i> | 1-435 |
| device. write | file marks to SCSI sequential | <i>scsiWrtFileMarks()</i> | 2-701 |
| find module by | file name and path. | <i>moduleFindByNameAndPath()</i> | 2-439 |
| generate temporary | file name (ANSI). | <i>tmpnam()</i> | 2-921 |
| /object module by specifying | file name or module ID. | <i>unld()</i> | 2-944 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|--------------------------|-------|
| standard input/output/error | FILE of current task. return | <i>stdioFp()</i> | 2-816 |
| read bytes from | file or device. | <i>read()</i> | 2-614 |
| test error indicator for | file pointer (ANSI). | <i>ferror()</i> | 2-199 |
| display | file pointer internals. | <i>stdioShow()</i> | 2-817 |
| stream/ store current value of | file position indicator for | <i>fgetpos()</i> | 2-200 |
| stream (ANSI). set | file position indicator for | <i>fseek()</i> | 2-226 |
| stream (ANSI). set | file position indicator for | <i>fsetpos()</i> | 2-227 |
| return current value of | file position indicator for/ | <i>ftell()</i> | 2-229 |
| beginning of file (ANSI). set | file position indicator to | <i>rewind()</i> | 2-631 |
| truncate | file (POSIX). | <i>ftruncate()</i> | 2-237 |
| delete | file (POSIX). | <i>unlink()</i> | 2-946 |
| set | file read/write pointer. | <i>lseek()</i> | 2-349 |
| open | file specified by fd (POSIX). | <i>fdopen()</i> | 2-195 |
| open | file specified by name (ANSI). | <i>fopen()</i> | 2-207 |
| open | file specified by name (ANSI). | <i>freopen()</i> | 2-221 |
| (POSIX). get | file status information | <i>fstat()</i> | 2-228 |
| (POSIX). get | file status information | <i>fstatfs()</i> | 2-229 |
| pathname (POSIX). get | file status information using | <i>stat()</i> | 2-815 |
| pathname (POSIX). get | file status information using | <i>statfs()</i> | 2-815 |
| asynchronous | file synchronization (POSIX). | <i>aiio_fsync()</i> | 2-10 |
| device and create dosFs | file system. initialize | <i>dosFsMkfs()</i> | 2-144 |
| mount NFS | file system. | <i>nfsMount()</i> | 2-530 |
| device and mount DOS | file system. initialize | <i>pccardMkfs()</i> | 2-552 |
| mount DOS | file system. | <i>pccardMount()</i> | 2-552 |
| device and create rt11Fs | file system. initialize | <i>rt11FsMkfs()</i> | 2-661 |
| set dosFs | file system date. | <i>dosFsDateSet()</i> | 2-140 |
| set rt11Fs | file system date. | <i>rt11FsDateSet()</i> | 2-658 |
| disk. mount DOS | file system from ATA hard | <i>usrAtaConfig()</i> | 2-947 |
| mount DOS | file system from floppy disk. | <i>usrFdConfig()</i> | 2-948 |
| disk. mount DOS | file system from IDE hard | <i>usrIdeConfig()</i> | 2-949 |
| exported file systems. remove | file system from list of | <i>nfsUnexport()</i> | 2-531 |
| /block device with dosFs | file system functions. | <i>dosFsDevInit()</i> | 2-141 |
| associate device with ntPassFs | file system functions. | <i>ntPassFsDevInit()</i> | 2-539 |
| associate device with passFs | file system functions. | <i>passFsDevInit()</i> | 2-549 |
| pass-through (to Windows NT) | file system library. | <i>ntPassFsLib</i> | 1-274 |
| raw block device | file system library. | <i>rawFsLib</i> | 1-299 |
| RT-11 media-compatible | file system library. | <i>rt11FsLib</i> | 1-311 |
| tape sequential device | file system library. | <i>tapeFsLib</i> | 1-380 |
| ISO 9660 CD-ROM read-only | file system library. | <i>cdromFsLib</i> | 1-52 |
| MS-DOS media-compatible | file system library. | <i>dosFsLib</i> | 1-82 |
| pass-through (to UNIX) | file system library (VxSim). | <i>passFsLib</i> | 1-279 |
| Network | File System (NFS) I/O driver. | <i>nfsDrv</i> | 1-268 |
| Network | File System (NFS) library. | <i>nfsLib</i> | 1-270 |
| library. Network | File System (NFS) server | <i>nfsdLib</i> | 1-266 |
| initialize | file system on block device. | <i>diskInit()</i> | 2-137 |
| set dosFs | file system time. | <i>dosFsTimeSet()</i> | 2-146 |
| exported. specify | file system to be NFS | <i>nfsExport()</i> | 2-528 |
| system from list of exported | file systems. remove file | <i>nfsUnexport()</i> | 2-531 |
| specified host. mount all | file systems exported by | <i>nfsMountAll()</i> | 2-531 |
| display exported | file systems of remote host. | <i>nfsExportShow()</i> | 2-528 |

| | Keyword | Name | Page |
|--|--|--------------------------------|--------|
| | put file to remote system. | <i>tftpPut()</i> | 2-905 |
| | library. File Transfer Protocol (FTP) | ftpLib | 1-121 |
| | server. File Transfer Protocol (FTP) | ftpdLib | 1-120 |
| | library. Trivial File Transfer Protocol server | tftpdLib | 1-392 |
| | client library. Trivial File Transfer Protocol (TFTP) | tftpLib | 1-393 |
| | create temporary binary file (Unimplemented) (ANSI). | <i>tmpfile()</i> | 2-920 |
| | transfer file via TFTP. | <i>tftpCopy()</i> | 2-899 |
| | interface. transfer file via TFTP using stream | <i>tftpXfer()</i> | 2-907 |
| | return very large float. | <i>infinityf()</i> | 2-274 |
| | probe for presence of floating-point coprocessor. | <i>fppProbe()</i> | 2-210 |
| | context. restore floating-point coprocessor | <i>fppRestore()</i> | 2-211 |
| | context. save floating-point coprocessor | <i>fppSave()</i> | 2-212 |
| | architecture-dependent floating-point coprocessor/ | fppArchLib | 1-118 |
| | support. initialize floating-point coprocessor | <i>fppInit()</i> | 2-210 |
| | support library. floating-point coprocessor | fppLib | 1-119 |
| | library. high-level floating-point emulation | mathSoftLib | 1-220 |
| | scanning library. floating-point formatting and | floatLib | 1-117 |
| | initialize floating-point I/O support. | <i>floatInit()</i> | 2-204 |
| | hardware floating-point math library. | mathHardLib | 1-220 |
| | initialize hardware floating-point math support. | <i>mathHardInit()</i> | 2-402 |
| | initialize software floating-point math support. | <i>mathSoftInit()</i> | 2-402 |
| | integer and fraction/ separate floating-point number into | <i>modf()</i> | 2-435 |
| | normalized fraction and/ break floating-point number into | <i>frexp()</i> | 2-221 |
| | print contents of task's floating-point registers. | <i>fppTaskRegsShow()</i> | 2-214 |
| | task TCB. get floating-point registers from | <i>fppTaskRegsGet()</i> | 2-213 |
| | task. set floating-point registers of | <i>fppTaskRegsSet()</i> | 2-214 |
| | initialize floating-point show facility. | <i>fppShowInit()</i> | 2-213 |
| | floating-point show routines. | fppShow | 1-120 |
| | initialize driver and/ publish fn network interface and | <i>fnattach()</i> | 2-207 |
| | portions of second to NTP format. convert | <i>sntpsNsecToFraction()</i> | 2-798 |
| | format disk. | <i>diskFormat()</i> | 2-136 |
| | convert format string. | <i>fioFormatV()</i> | 2-202 |
| | device. issue FORMAT_UNIT command to SCSI | <i>scsiFormatUnit()</i> | 2-677 |
| | find size of largest available free block. | <i>memPartFindMax()</i> | 2-424 |
| | find size of largest available free block. | <i>VXWMemPart::findMax()</i> | 2-989 |
| | system partition/ find largest free block in shared memory | <i>smMemFindMax()</i> | 2-760 |
| | partition. find largest free block in system memory | <i>memFindMax()</i> | 2-420 |
| | free block of memory. | <i>cfree()</i> | 2-83 |
| | free block of memory (ANSI). | <i>free()</i> | 2-220 |
| | partition. free block of memory in | <i>memPartFree()</i> | 2-424 |
| | partition. free block of memory in | <i>VXWMemPart::free()</i> | 2-989 |
| | cacheDmaMalloc(). free buffer acquired with | <i>cacheDmaFree()</i> | 2-54 |
| | determine number of free bytes in ring buffer. | <i>VXWRingBuf::freeBytes()</i> | 2-1010 |
| | determine number of free bytes in ring buffer. | <i>rngFreeBytes()</i> | 2-649 |
| | mBlk-clBlk-cluster/ free chain of | <i>netMblkClChainFree()</i> | 2-510 |
| | back to memory pool. free clBlk-cluster construct | <i>netClBlkFree()</i> | 2-504 |
| | pool. free cluster back to memory | <i>netClFree()</i> | 2-505 |
| | by SNMP master agent. free IPC resources allocated | <i>masterIpcFree()</i> | 2-399 |
| | free mBlk back to memory pool. | <i>netMblkFree()</i> | 2-513 |
| | construct. free mBlk-clBlk-cluster | <i>netMblkClFree()</i> | 2-510 |

| Keyword | Name | Page |
|--|--------------------------------------|--------------------------------------|
| OSPF tasks. | free OSPF resources and delete | <i>ospfTerminate()</i> 2-548 |
| SNMP master agent. | free resources allocated for | <i>masterQueCleanup()</i> 2-401 |
| partition block of memory/ reclaim fragmented | free shared memory system | <i>smMemFree()</i> 2-761 |
| | free space on RT-11 volume. | <i>squeeze()</i> 2-805 |
| | free specified IPC mechanism. | <i>salpcFree()</i> 2-665 |
| | free tuples from linked list. | <i>cisFree()</i> 2-86 |
| | free up list. | <i>lstFree()</i> 2-353 |
| | free up list. | <i>VXWList::~VXWList()</i> 2-987 |
| times out. | free up resources after query | <i>snmpMasterCleanup()</i> 2-780 |
| display meaning of specified | fsr value, symbolically/ | <i>fsrShow()</i> 2-227 |
| list directory contents via | FTP. | <i>ftpLs()</i> 2-234 |
| initiate transfer via | FTP. | <i>ftpXfer()</i> 2-235 |
| send | FTP command and get reply. | <i>ftpCommand()</i> 2-230 |
| get | FTP command reply. | <i>ftpReplyGet()</i> 2-235 |
| get completed | FTP data connection. | <i>ftpDataConnGet()</i> 2-231 |
| initialize | FTP data connection. | <i>ftpDataConnInit()</i> 2-231 |
| log in to remote | FTP server. | <i>ftpLogin()</i> 2-234 |
| get control connection to | FTP server on specified host. | <i>ftpHookup()</i> 2-233 |
| terminate | FTP server task. | <i>ftpdDelete()</i> 2-232 |
| initialize | FTP server task. | <i>ftpdInit()</i> 2-232 |
| initialize | Fujitsu MB86930 cache library. | <i>cacheMb930LibInit()</i> 2-67 |
| cache management library. | Fujitsu MB86930 (SPARClite) | <i>cacheMb930Lib</i> 1-46 |
| network interface/ END-style | Fujitsu MB86960 Ethernet | <i>mb86960End</i> 1-221 |
| network interface driver. | Fujitsu MB86960 NICE Ethernet | <i>if_fn</i> 1-151 |
| Controller (SPC) library. | Fujitsu MB87030 SCSI Protocol | <i>mb87030Lib</i> 1-223 |
| address (VxMP Opt.). convert | global address to local | <i>smObjGlobalToLocal()</i> 2-773 |
| convert local address to | global address (VxMP Opt.). | <i>smObjLocalToGlobal()</i> 2-776 |
| initialize | global mapping. | <i>vmBaseGlobalMapInit()</i> 2-957 |
| initialize | global mapping (VxVMI Opt.). | <i>vmGlobalMapInit()</i> 2-964 |
| get fd for | global standard/ | <i>ioGlobalStdGet()</i> 2-290 |
| set fd for | global standard/ | <i>ioGlobalStdSet()</i> 2-290 |
| initialize | global state for MUX. | <i>muxLibInit()</i> 2-477 |
| get MIB-II ICMP-group | global variables. | <i>m2IcmpGroupInfoGet()</i> 2-359 |
| /to virtual space in shared | global virtual mem (VxVMI/ | <i>vmGlobalMap()</i> 2-963 |
| information (VxVMI Opt.). get | global virtual memory | <i>vmGlobalInfoGet()</i> 2-963 |
| environment/ perform non-local | goto by restoring saved | <i>longjmp()</i> 2-346 |
| initialize system | hardware. | <i>sysHwInit()</i> 2-848 |
| set | hardware breakpoint. | <i>bh()</i> 2-34 |
| library. | hardware floating-point math | <i>mathHardLib</i> 1-220 |
| support. initialize | hardware floating-point math | <i>mathHardInit()</i> 2-402 |
| connect C routine to | hardware interrupt. | <i>intConnect()</i> 2-275 |
| disabled. inform SCSI that | hardware snooping of caches is | <i>scsiCacheSnoopDisable()</i> 2-675 |
| enabled. inform SCSI that | hardware snooping of caches is | <i>scsiCacheSnoopEnable()</i> 2-675 |
| display debugging | help menu. | <i>dbgHelp()</i> 2-109 |
| display NFS | help menu. | <i>nfsHelp()</i> 2-529 |
| display task monitoring | help menu. | <i>spyHelp()</i> 2-802 |
| test whether character is | hexadecimal digit (ANSI). | <i>isxdigit()</i> 2-309 |
| display or set size of shell | history. | <i>h()</i> 2-246 |
| display or set size of shell | history. | <i>shellHistory()</i> 2-739 |
| sample authentication | hook. | <i>ripAuthHook()</i> 2-633 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|-----------------------------------|--------|
| initialize task | hook facilities. | <i>taskHookInit()</i> | 2-870 |
| permanent address storage | hook for server. assign | <i>dhcpsAddressHookAdd()</i> | 2-132 |
| assign permanent lease storage | hook for server. | <i>dhcpsLeaseHookAdd()</i> | 2-135 |
| remove authentication | hook from RIP interface. | <i>ripAuthHookDelete()</i> | 2-636 |
| remove table bypass | hook from RIP interface. | <i>ripLeakHookDelete()</i> | 2-639 |
| PPP | hook library. | <i>pppHookLib</i> | 1-292 |
| task | hook library. | <i>taskHookLib</i> | 1-384 |
| delete network interface input | hook routine. | <i>etherInputHookDelete()</i> | 2-177 |
| network interface output | hook routine. delete | <i>etherOutputHookDelete()</i> | 2-181 |
| delete ip filter | hook routine. | <i>ipFilterHookDelete()</i> | 2-301 |
| previously added module create | hook routine. delete | <i>moduleCreateHookDelete()</i> | 2-437 |
| add | hook routine on unit basis. | <i>pppHookAdd()</i> | 2-577 |
| delete | hook routine on unit basis. | <i>pppHookDelete()</i> | 2-578 |
| initialize task | hook show facility. | <i>taskHookShowInit()</i> | 2-870 |
| task | hook show routines. | <i>taskHookShow</i> | 1-386 |
| routing tables. add | hook to bypass RIP and kernel | <i>ripLeakHookAdd()</i> | 2-639 |
| add authentication | hook to RIP interface. | <i>ripAuthHookAdd()</i> | 2-633 |
| Ethernet raw I/O routines and | hooks. | <i>etherLib</i> | 1-109 |
| ip filter | hooks library. | <i>ipFilterLib</i> | 1-181 |
| address. get local address | (host number) from Internet | <i>inet_1naof()</i> | 2-269 |
| address from network and | host numbers. form Internet | <i>inet_makeaddr()</i> | 2-270 |
| address from network and | host numbers. form Internet | <i>inet_makeaddr_b()</i> | 2-270 |
| add host to | host table. | <i>hostAdd()</i> | 2-249 |
| delete host from | host table. | <i>hostDelete()</i> | 2-250 |
| display | host table. | <i>hostShow()</i> | 2-251 |
| initialize network | host table. | <i>hostTblInit()</i> | 2-251 |
| address. look up host in | host table by Internet | <i>hostGetByAddr()</i> | 2-250 |
| look up host in | host table by name. | <i>hostGetByName()</i> | 2-251 |
| | host table subroutine library. | <i>hostLib</i> | 1-123 |
| compute | hyperbolic cosine (ANSI). | <i>cosh()</i> | 2-95 |
| compute | hyperbolic cosine (ANSI). | <i>coshf()</i> | 2-96 |
| compute | hyperbolic sine (ANSI). | <i>sinh()</i> | 2-755 |
| compute | hyperbolic sine (ANSI). | <i>sinhf()</i> | 2-755 |
| compute | hyperbolic tangent (ANSI). | <i>tanh()</i> | 2-861 |
| compute | hyperbolic tangent (ANSI). | <i>tanhf()</i> | 2-862 |
| register (MC680x0, MIPS, | i386/i486). set task status | <i>VXWTask::SRSet()</i> | 2-1037 |
| register edi (also esi - eax) | (i386/i486). /contents of | <i>edi()</i> | 2-150 |
| contents of status register | (i386/i486). return | <i>eflags()</i> | 2-152 |
| register (MC680x0, MIPS, | i386/i486). set task status | <i>taskSRSet()</i> | 2-886 |
| | I8250 serial driver. | <i>i8250Sio</i> | 1-123 |
| return contents of register fp | (i960). | <i>fp()</i> | 2-209 |
| g1 - g7 (SPARC) and g1 - g14 | (i960). /of register g0, also | <i>g0()</i> | 2-238 |
| contents of acw register | (i960). return | <i>acw()</i> | 2-4 |
| contents of pcw register | (i960). return | <i>pcw()</i> | 2-556 |
| contents of register pfp | (i960). return | <i>pfp()</i> | 2-569 |
| I960Cx 1KB instruction cache | (i960). load and lock | <i>cacheI960CxIC1kLoadNLock()</i> | 2-58 |
| I960Cx instruction cache | (i960). disable | <i>cacheI960CxICDisable()</i> | 2-58 |
| I960Cx instruction cache | (i960). enable | <i>cacheI960CxICEnable()</i> | 2-58 |
| I960Cx instruction cache | (i960). invalidate | <i>cacheI960CxICInvalidate()</i> | 2-59 |
| 512-byte instruction cache | (i960). load and lock I960Cx | <i>cacheI960CxICLoadNLock()</i> | 2-59 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|---------------------------------|-------|
| of register r3 (also r4 - r15) | (i960). return contents | r3() | 2-607 |
| I960Cx cache library | (i960). initialize | cacheI960CxLibInit() | 2-60 |
| ensure data cache coherency | (i960). | cacheI960JxDCCoherent() | 2-60 |
| disable I960Jx data cache | (i960). | cacheI960JxDCDisable() | 2-60 |
| enable I960Jx data cache | (i960). | cacheI960JxDCEnable() | 2-61 |
| flush I960Jx data cache | (i960). | cacheI960JxDCFlush() | 2-61 |
| invalidate I960Jx data cache | (i960). | cacheI960JxDCInvalidate() | 2-61 |
| get I960Jx data cache status | (i960). | cacheI960JxDCStatusGet() | 2-62 |
| I960Jx instruction cache | (i960). disable | cacheI960JxICDisable() | 2-62 |
| I960Jx instruction cache | (i960). enable | cacheI960JxICEnable() | 2-62 |
| contents of register rip | (i960). return | rip() | 2-632 |
| flush I960Jx instruction cache | (i960). | cacheI960JxICFlush() | 2-63 |
| I960Jx instruction cache | (i960). invalidate | cacheI960JxICInvalidate() | 2-63 |
| lock I960Jx instruction cache | (i960). load and | cacheI960JxICLoadNLock() | 2-63 |
| I960Jx I-cache locking status | (i960). get | cacheI960JxICLockingStatusGet() | 2-64 |
| instruction cache status | (i960). get I960Jx | cacheI960JxICStatusGet() | 2-64 |
| I960Jx cache library | (i960). initialize | cacheI960JxLibInit() | 2-64 |
| contents of tcw register | (i960). return | tcw() | 2-897 |
| return contents of register sp | (i960). | tsp() | 2-924 |
| level (MC680x0, SPARC, | i960, x86, ARM). /interrupt | intLevelSet() | 2-279 |
| level (MC680x0, SPARC, | i960, x86, ARM). /lock-out | intLockLevelGet() | 2-282 |
| level (MC680x0, SPARC, | i960, x86, ARM). /lock-out | intLockLevelSet() | 2-282 |
| vector table (MC680x0, SPARC, | i960, x86, ARM). /exception | intVecTableWriteProtect() | 2-288 |
| for C routine (MC680x0, SPARC, | i960, x86, MIPS). /handler | intHandlerCreate() | 2-279 |
| vector (MC680x0, SPARC, | i960, x86, MIPS). /interrupt | intVecGet() | 2-286 |
| vector (trap) (MC680x0, SPARC, | i960, x86, MIPS). set CPU | intVecSet() | 2-286 |
| base address (MC680x0, SPARC, | i960, x86, MIPS, ARM). /(trap) | intVecBaseGet() | 2-284 |
| base address (MC680x0, SPARC, | i960, x86, MIPS, ARM). /(trap) | intVecBaseSet() | 2-285 |
| (i960). load and lock | I960Cx 1KB instruction cache | cacheI960CxIC1kLoadNLock() | 2-58 |
| cache (i960). load and lock | I960Cx 512-byte instruction | cacheI960CxICLoadNLock() | 2-59 |
| initialize | I960Cx cache library (i960). | cacheI960CxLibInit() | 2-60 |
| assembly routines. | I960Cx cache management | cacheI960CxALib | 1-35 |
| library. | I960Cx cache management | cacheI960CxLib | 1-36 |
| (i960). disable | I960Cx instruction cache | cacheI960CxICDisable() | 2-58 |
| (i960). enable | I960Cx instruction cache | cacheI960CxICEnable() | 2-58 |
| (i960). invalidate | I960Cx instruction cache | cacheI960CxICInvalidate() | 2-59 |
| initialize | I960Jx cache library (i960). | cacheI960JxLibInit() | 2-64 |
| assembly routines. | I960Jx cache management | cacheI960JxALib | 1-36 |
| library. | I960Jx cache management | cacheI960JxLib | 1-37 |
| disable | I960Jx data cache (i960). | cacheI960JxDCDisable() | 2-60 |
| enable | I960Jx data cache (i960). | cacheI960JxDCEnable() | 2-61 |
| flush | I960Jx data cache (i960). | cacheI960JxDCFlush() | 2-61 |
| invalidate | I960Jx data cache (i960). | cacheI960JxDCInvalidate() | 2-61 |
| (i960). get | I960Jx data cache status | cacheI960JxDCStatusGet() | 2-62 |
| (i960). get | I960Jx I-cache locking status . | cacheI960JxICLockingStatusGet() | 2-64 |
| (i960). disable | I960Jx instruction cache | cacheI960JxICDisable() | 2-62 |
| (i960). enable | I960Jx instruction cache | cacheI960JxICEnable() | 2-62 |
| (i960). flush | I960Jx instruction cache | cacheI960JxICFlush() | 2-63 |
| (i960). invalidate | I960Jx instruction cache | cacheI960JxICInvalidate() | 2-63 |
| (i960). load and lock | I960Jx instruction cache | cacheI960JxICLoadNLock() | 2-63 |

| | Keyword | Name | Page |
|------------------------------|--------------------------------------|---------------------------------|--------|
| status (i960). get | I960Jx instruction cache | <i>cacheI960JxICStatusGet()</i> | 2-64 |
| display statistics for | ICMP. | <i>icmpstatShow()</i> | 2-255 |
| all resources used to access | ICMP group. delete | <i>m2IcmpDelete()</i> | 2-359 |
| routines. | ICMP information display | <i>icmpShow</i> | 1-124 |
| initialize | ICMP show routines. | <i>icmpShowInit()</i> | 2-255 |
| create device for | IDE disk. | <i>ideDevCreate()</i> | 2-256 |
| | IDE disk device driver. | <i>ideDrv</i> | 1-125 |
| initialize | IDE driver. | <i>ideDrv()</i> | 2-256 |
| mount DOS file system from | IDE hard disk. | <i>usrIdeConfig()</i> | 2-949 |
| | IDE initialization. | <i>usrIde</i> | 1-411 |
| header block and demuxer | information. create | <i>hdrBlkBuild()</i> | 2-246 |
| get partition | information. | <i>memPartInfoGet()</i> | 2-425 |
| get PPP link status | information. | <i>pppInfoGet()</i> | 2-579 |
| display PPP link status | information. | <i>pppInfoShow()</i> | 2-579 |
| show volume configuration | information. | <i>cdromFsVolConfigShow()</i> | 2-82 |
| show CIS | information. | <i>cisShow()</i> | 2-87 |
| get TFTP status | information. | <i>tftpInfoShow()</i> | 2-903 |
| print VxWorks version | information. | <i>version()</i> | 2-955 |
| get partition | information. | <i>VXWMemPart::info()</i> | 2-990 |
| queue. get | information about message | <i>VXWMsgQ::info()</i> | 2-1002 |
| queue. show | information about message | <i>VXWMsgQ::show()</i> | 2-1007 |
| queue. get | information about message | <i>msgQInfoGet()</i> | 2-460 |
| queue. show | information about message | <i>msgQShow()</i> | 2-464 |
| object module. get | information about | <i>moduleInfoGet()</i> | 2-441 |
| object module. get | information about | <i>VXWModule::info()</i> | 2-993 |
| show | information about semaphore. | <i>VXWSem::show()</i> | 2-1016 |
| show | information about semaphore. | <i>semShow()</i> | 2-717 |
| memory network. show | information about shared | <i>smNetShow()</i> | 2-772 |
| get | information about task. | <i>VXWTask::info()</i> | 2-1029 |
| get | information about task. | <i>taskInfoGet()</i> | 2-873 |
| show | information about watchdog. | <i>wdShow()</i> | 2-1062 |
| ICMP | Information display routines. | <i>icmpShow</i> | 1-124 |
| IGMP | information display routines. | <i>igmpShow</i> | 1-171 |
| network | information display routines. | <i>netShow</i> | 1-265 |
| TCP | information display routines. | <i>tcpShow</i> | 1-391 |
| UDP | information display routines. | <i>udpShow</i> | 1-403 |
| DHCP run-time client | information display routines. | <i>dhcpcShow</i> | 1-74 |
| device. show status | information for physical | <i>scsiPhysDevShow()</i> | 2-688 |
| show status | information for SCSI manager. | <i>scsiMgrShow()</i> | 2-682 |
| display debugging | information for TCP protocol. | <i>tcpDebugShow()</i> | 2-896 |
| address. extract lease | information from Internet | <i>bootLeaseExtract()</i> | 2-37 |
| get addressing | information from packet. | <i>muxPacketAddrGet()</i> | 2-479 |
| CIS. get | information from PC card's | <i>cisGet()</i> | 2-86 |
| device. read configuration | information from requested NFS | <i>nfsDevInfoGet()</i> | 2-524 |
| print complete | information from task's TCB. | <i>ti()</i> | 2-908 |
| display task | information from TCBs. | <i>VXWTask::show()</i> | 2-1036 |
| display task | information from TCBs. | <i>taskShow()</i> | 2-882 |
| task | information library. | <i>taskInfo</i> | 1-386 |
| object. print | information on specified | <i>show()</i> | 2-741 |
| get file status | information (POSIX). | <i>fstat()</i> | 2-228 |

| | Keyword | Name | Page |
|--|--|-----------------------------------|-------|
| | get file status information (POSIX). | <i>fstatfs()</i> | 2-229 |
| | and v2 library. Routing Information Protocol (RIP) v1 | <i>ripLib</i> | 1-306 |
| | device. send control information to MUX or to | <i>muxIoctl()</i> | 2-476 |
| | reclaim/ delete module ID information (use <i>unld()</i> to) | <i>moduleDelete()</i> | 2-438 |
| | (POSIX). get file status information using pathname | <i>stat()</i> | 2-815 |
| | (POSIX). get file status information using pathname | <i>statfs()</i> | 2-815 |
| | get global virtual memory information (VxVMI Opt.). | <i>vmGlobalInfoGet()</i> | 2-963 |
| | ATA initialization. | <i>usrAta</i> | 1-409 |
| | floppy disk initialization. | <i>usrFd</i> | 1-410 |
| | IDE initialization. | <i>usrIde</i> | 1-411 |
| | SCSI initialization. | <i>usrScsi</i> | 1-413 |
| | shared memory object initialization. | <i>usrSmObj</i> | 1-413 |
| | DHCP client library initialization. | <i>dhcpcLibInit()</i> | 2-123 |
| | generic ROM initialization. | <i>romStart()</i> | 2-652 |
| | perform generic SCSI thread initialization. | <i>scsiThreadInit()</i> | 2-700 |
| | ROM initialization module. | <i>bootInit</i> | 1-29 |
| | user-defined system initialization routine. | <i>usrInit()</i> | 2-950 |
| | ROM MMU initialization (SPARC). | <i>mmuSparcLib</i> | 1-237 |
| | parse initialization string. | <i>el3c90xInitParse()</i> | 2-159 |
| | parse initialization string. | <i>ln97xInitParse()</i> | 2-321 |
| | parse initialization string. | <i>ln7990InitParse()</i> | 2-324 |
| | parse initialization string. | <i>mb86960InitParse()</i> | 2-404 |
| | parse initialization string. | <i>nicEvbInitParse()</i> | 2-533 |
| | get fd for global standard input/output/error. | <i>ioGlobalStdGet()</i> | 2-290 |
| | set fd for global standard input/output/error. | <i>ioGlobalStdSet()</i> | 2-290 |
| | get fd for task standard input/output/error. | <i>ioTaskStdGet()</i> | 2-298 |
| | set fd for task standard input/output/error. | <i>ioTaskStdSet()</i> | 2-298 |
| | set shell's default input/output/error fds. | <i>shellOrigStdSet()</i> | 2-740 |
| | current task. return standard input/output/error FILE of | <i>stdioFp()</i> | 2-816 |
| | device. issue INQUIRY command to SCSI | <i>scsiInquiry()</i> | 2-679 |
| | MTRR table with WRMSR instruction. /from specified | <i>pentiumMtrrSet()</i> | 2-561 |
| | synchronize instruction and data caches. | <i>cacheTextUpdate()</i> | 2-74 |
| | return size of R3000 instruction cache. | <i>cacheR3kIsize()</i> | 2-69 |
| | load and lock I960Cx 1KB instruction cache (i960). | <i>cacheI960CxIC1kLoadNLock()</i> | 2-58 |
| | disable I960Cx instruction cache (i960). | <i>cacheI960CxICDisable()</i> | 2-58 |
| | enable I960Cx instruction cache (i960). | <i>cacheI960CxICEnable()</i> | 2-58 |
| | invalidate I960Cx instruction cache (i960). | <i>cacheI960CxICInvalidate()</i> | 2-59 |
| | load and lock I960Cx 512-byte instruction cache (i960). | <i>cacheI960CxICLoadNLock()</i> | 2-59 |
| | disable I960Jx instruction cache (i960). | <i>cacheI960JxICDisable()</i> | 2-62 |
| | enable I960Jx instruction cache (i960). | <i>cacheI960JxICEnable()</i> | 2-62 |
| | flush I960Jx instruction cache (i960). | <i>cacheI960JxICFlush()</i> | 2-63 |
| | invalidate I960Jx instruction cache (i960). | <i>cacheI960JxICInvalidate()</i> | 2-63 |
| | load and lock I960Jx instruction cache (i960). | <i>cacheI960JxICLoadNLock()</i> | 2-63 |
| | (i960). get I960Jx instruction cache status | <i>cacheI960JxICStatusGet()</i> | 2-64 |
| | execute serializing instruction CPUID. | <i>pentiumSerialize()</i> | 2-565 |
| | /atomic compare-and-exchange instruction to clear bit. | <i>pentiumBtc()</i> | 2-557 |
| | /atomic compare-and-exchange instruction to set bit. | <i>pentiumBts()</i> | 2-557 |
| | display specified number of instructions. disassemble and | <i>I()</i> | 2-312 |
| | convert string to int (ANSI). | <i>atoi()</i> | 2-26 |
| | Internet address to long integer. convert dot notation | <i>inet_addr()</i> | 2-268 |

| | Keyword | Name | Page |
|--------------------------------|---|-----------------------------|--------|
| double-precision value to | integer. convert | <i>irint()</i> | 2-302 |
| single-precision value to | integer. convert | <i>irintf()</i> | 2-302 |
| round number to nearest | integer. | <i>iround()</i> | 2-303 |
| round number to nearest | integer. | <i>iroundf()</i> | 2-303 |
| round number to nearest | integer. | <i>round()</i> | 2-652 |
| round number to nearest | integer. | <i>roundf()</i> | 2-653 |
| truncate to | integer. | <i>trunc()</i> | 2-923 |
| truncate to | integer. | <i>truncf()</i> | 2-923 |
| /floating-point number into | integer and fraction parts/ | <i>modf()</i> | 2-435 |
| compute absolute value of | integer (ANSI). | <i>abs()</i> | 2-2 |
| convert string to long | integer (ANSI). | <i>strtol()</i> | 2-830 |
| string to unsigned long | integer (ANSI). convert | <i>strtoul()</i> | 2-831 |
| generate pseudo-random | integer between 0 and RAND_MAX / | <i>rand()</i> | 2-610 |
| read next word (32-bit | integer) from stream. | <i>getw()</i> | 2-244 |
| to specified/ compute smallest | integer greater than or equal | <i>ceil()</i> | 2-82 |
| to specified/ compute smallest | integer greater than or equal | <i>ceilf()</i> | 2-83 |
| specified/ compute largest | integer less than or equal to | <i>floor()</i> | 2-205 |
| specified/ compute largest | integer less than or equal to | <i>floorf()</i> | 2-205 |
| write word (32-bit | integer) to stream. | <i>putw()</i> | 2-605 |
| adaptor chip library. | Intel 82365SSL PCMCIA host bus | <i>pcic</i> | 1-280 |
| adaptor chip show library. | Intel 82365SSL PCMCIA host bus | <i>pcicShow</i> | 1-281 |
| interface driver. END style | Intel 82557 Ethernet network | <i>fei82557End</i> | 1-114 |
| interface driver. | Intel 82557 Ethernet network | <i>if_fei</i> | 1-149 |
| interface driver. | Intel 82596 Ethernet network | <i>if_ei</i> | 1-137 |
| interface driver. END style | Intel 82596 Ethernet network | <i>ei82596End</i> | 1-96 |
| interface driver for hkv3500. | Intel 82596 Ethernet network | <i>if_eihk</i> | 1-140 |
| interface driver. | Intel EtherExpress 16 network | <i>if_eex</i> | 1-136 |
| interface driver. END style | Intel Olicom PCMCIA network | <i>iOlicomEnd</i> | 1-177 |
| Ethernet address for specified | Internet address. resolve | <i>etherAddrResolve()</i> | 2-175 |
| look up host in host table by | Internet address. | <i>hostGetByAddr()</i> | 2-250 |
| address (host number) from | Internet address. get local | <i>inet_lnaof()</i> | 2-269 |
| return network number from | Internet address. | <i>inet_netof()</i> | 2-271 |
| extract lease information from | Internet address. | <i>bootLeaseExtract()</i> | 2-37 |
| extract net mask field from | Internet address. | <i>bootNetmaskExtract()</i> | 2-38 |
| and host numbers. form | Internet address from network | <i>inet_makeaddr()</i> | 2-270 |
| and host numbers. form | Internet address from network | <i>inet_makeaddr_b()</i> | 2-270 |
| routines. | Internet address manipulation | <i>inetLib</i> | 1-172 |
| interface. get | Internet address of network | <i>ifAddrGet()</i> | 2-258 |
| point-to-point peer. get | Internet address of | <i>ifDstAddrGet()</i> | 2-260 |
| integer. convert dot notation | Internet address to long | <i>inet_addr()</i> | 2-268 |
| library. Packet | InterNet Grouper (PING) | <i>pingLib</i> | 1-288 |
| string to address. convert | Internet network number from | <i>inet_network()</i> | 2-272 |
| add routine to receive all | internet protocol packets. | <i>ipFilterHookAdd()</i> | 2-300 |
| /all active connections for | Internet protocol sockets. | <i>inetstatShow()</i> | 2-274 |
| channel's receive-character | interrupt. handle | <i>winIntRcv()</i> | 2-1065 |
| handle reciever | interrupt. | <i>z8530IntRd()</i> | 2-1069 |
| handle transmitter | interrupt. | <i>z8530IntWr()</i> | 2-1069 |
| handle receiver | interrupt. | <i>ambaIntRx()</i> | 2-14 |
| handle transmitter | interrupt. | <i>ambaIntTx()</i> | 2-14 |
| handle receiver/transmitter | interrupt. | <i>i8250Int()</i> | 2-254 |

| | Keyword | Name | Page |
|-------------------------------|--------------------------------------|-----------------------------|--------|
| connect C routine to hardware | interrupt. | <i>intConnect()</i> | 2-275 |
| handle SCC | interrupt. | <i>m68332Int()</i> | 2-387 |
| handle SCC | interrupt. | <i>m68360Int()</i> | 2-387 |
| handle receiver | interrupt. | <i>m68562RxInt()</i> | 2-388 |
| receiver/transmitter error | interrupt. handle | <i>m68562RxTxErrInt()</i> | 2-389 |
| handle transmitter | interrupt. | <i>m68562TxInt()</i> | 2-389 |
| handle receiver | interrupt. | <i>ns16550IntRd()</i> | 2-536 |
| handle transmitter | interrupt. | <i>ns16550IntWr()</i> | 2-537 |
| handle controller | interrupt. | <i>ntInt()</i> | 2-537 |
| handle receiver | interrupt. | <i>ppc403IntRd()</i> | 2-575 |
| handle transmitter | interrupt. | <i>ppc403IntWr()</i> | 2-576 |
| handle SMC | interrupt. | <i>ppc860Int()</i> | 2-576 |
| handle | interrupt. | <i>sa1100Int()</i> | 2-664 |
| handle receiver | interrupt. | <i>st16552IntRd()</i> | 2-813 |
| handle transmitter | interrupt. | <i>st16552IntWr()</i> | 2-814 |
| routine to auxiliary clock | interrupt. connect | <i>sysAuxClkConnect()</i> | 2-841 |
| acknowledge bus | interrupt. | <i>sysBusIntAck()</i> | 2-844 |
| generate bus | interrupt. | <i>sysBusIntGen()</i> | 2-844 |
| routine to system clock | interrupt. connect | <i>sysClkConnect()</i> | 2-846 |
| connect routine to mailbox | interrupt. | <i>sysMailboxConnect()</i> | 2-850 |
| enable mailbox | interrupt. | <i>sysMailboxEnable()</i> | 2-851 |
| buffer after data store error | interrupt. clean up store | <i>cleanUpStoreBuffer()</i> | 2-87 |
| channel's receive-character | interrupt.. handle | <i>unixIntRcv()</i> | 2-944 |
| ARM). disable corresponding | interrupt bits (MIPS, PowerPC, | <i>intDisable()</i> | 2-278 |
| ARM). enable corresponding | interrupt bits (MIPS, PowerPC, | <i>intEnable()</i> | 2-278 |
| handle receiver/transmitter | interrupt for NS 16550 chip. | <i>evbNs16550Int()</i> | 2-183 |
| network interface | interrupt handler. | <i>mbcIntr()</i> | 2-410 |
| routine (MC680x0,/ construct | interrupt handler for C | <i>intHandlerCreate()</i> | 2-279 |
| disable bus | interrupt level. | <i>sysIntDisable()</i> | 2-849 |
| enable bus | interrupt level. | <i>sysIntEnable()</i> | 2-849 |
| SPARC, i960, x86, ARM). set | interrupt level (MC680x0, | <i>intLevelSet()</i> | 2-279 |
| | interrupt level processing. | <i>ns16550Int()</i> | 2-535 |
| | interrupt level processing. | <i>st16552Int()</i> | 2-812 |
| multiplexed | interrupt level processing. | <i>st16552MuxInt()</i> | 2-814 |
| architecture-dependent | interrupt library. | <i>intArchLib</i> | 1-174 |
| (MC680x0, SPARC,/ get current | interrupt lock-out level | <i>intLockLevelGet()</i> | 2-282 |
| (MC680x0, SPARC,/ set current | interrupt lock-out level | <i>intLockLevelSet()</i> | 2-282 |
| cancel | interrupt locks. | <i>intUnlock()</i> | 2-284 |
| get current | interrupt nesting depth. | <i>intCount()</i> | 2-277 |
| /if current state is in | interrupt or task context. | <i>intContext()</i> | 2-276 |
| miscellaneous | interrupt processing. | <i>ns16550IntEx()</i> | 2-536 |
| miscellaneous | interrupt processing. | <i>st16552IntEx()</i> | 2-813 |
| user-defined system clock | interrupt routine. | <i>usrClock()</i> | 2-948 |
| interrupts. | interrupt service for card | <i>iOlicomIntHandle()</i> | 2-291 |
| architecture-independent | interrupt subroutine library. | <i>intLib</i> | 1-175 |
| SPARC, i960, x86, MIPS). get | interrupt vector (MC680x0, | <i>intVecGet()</i> | 2-286 |
| connect C routine to critical | interrupt vector (PowerPC/ | <i>exclIntCrtConnect()</i> | 2-187 |
| | interrupt-level input. | <i>tyIRd()</i> | 2-930 |
| | interrupt-level output. | <i>tyITx()</i> | 2-931 |
| handle error | interrupts. | <i>z8530IntEx()</i> | 2-1068 |

| | Keyword | Name | Page |
|---------------------------------|-----------------------------------|-----------------------------------|--------|
| | lock out | interrupts. <i>intLock()</i> | 2-280 |
| interrupt service for card | interrupts. | <i>iOlicomIntHandle()</i> | 2-291 |
| handle error | interrupts. | <i>ppc403IntEx()</i> | 2-575 |
| handle special status | interrupts. | <i>cd2400Int()</i> | 2-80 |
| handle receiver | interrupts. | <i>cd2400IntRx()</i> | 2-80 |
| handle transmitter | interrupts. | <i>cd2400IntTx()</i> | 2-80 |
| turn off auxiliary clock | interrupts. | <i>sysAuxClkDisable()</i> | 2-841 |
| turn on auxiliary clock | interrupts. | <i>sysAuxClkEnable()</i> | 2-842 |
| turn off system clock | interrupts. | <i>sysClkDisable()</i> | 2-846 |
| turn on system clock | interrupts. | <i>sysClkEnable()</i> | 2-847 |
| connect BSP serial device | interrupts. | <i>sysSerialHwInit2()</i> | 2-858 |
| enable | interrupts. | <i>unixDevInit2()</i> | 2-942 |
| entry point for handling | interrupts from 82596. | <i>eiInt()</i> | 2-156 |
| handle all | interrupts in one vector. | <i>z8530Int()</i> | 2-1068 |
| handle all DUART | interrupts in one vector. | <i>m68681Int()</i> | 2-393 |
| provide raw | I/O access. | <i>fdRawio()</i> | 2-196 |
| do raw | I/O access. | <i>ataRawio()</i> | 2-24 |
| provide raw | I/O access. | <i>ideRawio()</i> | 2-257 |
| initialize asynchronous | I/O (AIO) library. | <i>aioPxLibInit()</i> | 2-8 |
| asynchronous | I/O (AIO) library (POSIX). | <i>aioPxLib</i> | 1-4 |
| asynchronous | I/O (AIO) show library. | <i>aioPxShow</i> | 1-8 |
| perform | I/O control function. | <i>ioctl()</i> | 2-288 |
| perform device-specific | I/O control function. | <i>scsiIoctl()</i> | 2-680 |
| sequential access/ perform | I/O control function for | <i>scsiSeqIoctl()</i> | 2-694 |
| find | I/O device in device list. | <i>iosDevFind()</i> | 2-294 |
| microSparc I/II | I/O DMA library. | <i>ioMmuMicroSparcLib</i> | 1-180 |
| /L64862 MBus-to-SBus Interface: | I/O DMA library (SPARC). | <i>mmuL64862Lib</i> | 1-232 |
| network remote file | I/O driver. | <i>netDrv</i> | 1-263 |
| Network File System (NFS) | I/O driver. | <i>nfsDrv</i> | 1-268 |
| pipe | I/O driver. | <i>pipeDrv</i> | 1-289 |
| install | I/O driver. | <i>iosDrvInstall()</i> | 2-295 |
| remove | I/O driver. | <i>iosDrvRemove()</i> | 2-295 |
| virtual generic file | I/O driver for WDB agent. | <i>wdbTsfsDrv</i> | 1-435 |
| virtual tty | I/O driver for WDB agent. | <i>wdbVioDrv</i> | 1-440 |
| | I/O interface library. | <i>ioLib</i> | 1-176 |
| formatted | I/O library. | <i>fiolib</i> | 1-116 |
| initialize microSparc I/II | I/O MMU data structures. | <i>ioMmuMicroSparcInit()</i> | 2-292 |
| (SPARC). initialize L64862 | I/O MMU DMA data structures | <i>mmuL64862DmaInit()</i> | 2-433 |
| (TMS390S10/MB86904). map | I/O MMU for microSparc I/II | <i>ioMmuMicroSparcMap()</i> | 2-292 |
| /return status of asynchronous | I/O operation (POSIX). | <i>aio_return()</i> | 2-11 |
| /error status of asynchronous | I/O operation (POSIX). | <i>aio_error()</i> | 2-9 |
| (SCSI-1). NCR 53C710 SCSI | I/O Processor (SIOP) library | <i>ncr710Lib</i> | 1-255 |
| (SCSI-2). NCR 53C710 SCSI | I/O Processor (SIOP) library | <i>ncr710Lib2</i> | 1-256 |
| (SCSI-2). NCR 53C8xx PCI SCSI | I/O Processor (SIOP) library | <i>ncr810Lib</i> | 1-257 |
| wait for asynchronous | I/O request(s) (POSIX). | <i>aio_suspend()</i> | 2-12 |
| initiate list of asynchronous | I/O requests (POSIX). | <i>lio_listio()</i> | 2-318 |
| Ethernet raw | I/O routines and hooks. | <i>etherLib</i> | 1-109 |
| initialize standard | I/O show facility. | <i>stdioShowInit()</i> | 2-817 |
| initialize floating-point | I/O support. | <i>floatInit()</i> | 2-204 |
| initialize standard | I/O support. | <i>stdioInit()</i> | 2-816 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|-----------------------------|-------|
| initialize formatted | I/O support library. | <i>fioLibInit()</i> | 2-203 |
| add device to | I/O system. | <i>iosDevAdd()</i> | 2-293 |
| delete device from | I/O system. | <i>iosDevDelete()</i> | 2-293 |
| initialize | I/O system. | <i>iosInit()</i> | 2-297 |
| | I/O system library. | <i>iosLib</i> | 1-180 |
| initialize | I/O system show facility. | <i>iosShowInit()</i> | 2-297 |
| | I/O system show routines. | <i>iosShow</i> | 1-181 |
| initialize | kernel. | <i>kernellInit()</i> | 2-310 |
| announce clock tick to | kernel. | <i>tickAnnounce()</i> | 2-909 |
| /MB86930 automatic locking of | kernel instructions/data. | <i>cacheMb930LockAuto()</i> | 2-67 |
| VxWorks | kernel library. | <i>kernelLib</i> | 1-182 |
| return | kernel revision string. | <i>kernelVersion()</i> | 2-311 |
| add hook to bypass RIP and | kernel routing tables. | <i>ripLeakHookAdd()</i> | 2-639 |
| get value of | kernel's tick counter. | <i>tickGet()</i> | 2-909 |
| set value of | kernel's tick counter. | <i>tickSet()</i> | 2-910 |
| structures/ initialize | L64862 I/O MMU DMA data | <i>mmuL64862DmaInit()</i> | 2-433 |
| I/O DMA library/ LSI Logic | L64862 MBus-to-SBus Interface: | <i>mmuL64862Lib</i> | 1-232 |
| initialize memory for | Lance chip. | <i>ln7990InitMem()</i> | 2-323 |
| interface driver. AMD Am7990 | LANCE Ethernet network | <i>if_in</i> | 1-153 |
| interface/ END style AMD 7990 | LANCE Ethernet network | <i>ln7990End</i> | 1-190 |
| compare two strings | lexicographically (ANSI). | <i>strcmp()</i> | 2-819 |
| change | line-delete character. | <i>tyDeleteLineSet()</i> | 2-928 |
| read line with | line-editing. | <i>ledRead()</i> | 2-317 |
| | line-editing library. | <i>ledLib</i> | 1-184 |
| discard | line-editor ID. | <i>ledClose()</i> | 2-316 |
| create new | line-editor ID. | <i>ledOpen()</i> | 2-317 |
| change | line-editor ID parameters. | <i>ledControl()</i> | 2-316 |
| free tuples from | linked list. | <i>cisFree()</i> | 2-86 |
| simple | linked list class. | <i>VXWList</i> | 1-417 |
| library. doubly | linked list subroutine | <i>lstLib</i> | 1-198 |
| (C++). call all | linked static constructors | <i>cplusCtorsLink()</i> | 2-97 |
| (C++). call all | linked static destructors | <i>cplusDtorsLink()</i> | 2-99 |
| add option to option request | list. | <i>dhcpcBootOptionSet()</i> | 2-117 |
| add option to option request | list. | <i>dhcpcOptionSet()</i> | 2-125 |
| address to multicast address | list. add multicast | <i>etherMultiAdd()</i> | 2-178 |
| find I/O device in device | list. | <i>iosDevFind()</i> | 2-294 |
| add node to end of | list. | <i>lstAdd()</i> | 2-350 |
| report number of nodes in | list. | <i>lstCount()</i> | 2-351 |
| delete specified node from | list. | <i>lstDelete()</i> | 2-351 |
| extract sublist from | list. | <i>lstExtract()</i> | 2-352 |
| find node in | list. | <i>lstFind()</i> | 2-352 |
| find first node in | list. | <i>lstFirst()</i> | 2-353 |
| free up | list. | <i>lstFree()</i> | 2-353 |
| and return first node from | list. delete | <i>lstGet()</i> | 2-354 |
| find last node in | list. | <i>lstLast()</i> | 2-355 |
| find next node in | list. | <i>lstNext()</i> | 2-356 |
| find Nth node in | list. | <i>lstNth()</i> | 2-357 |
| find previous node in | list. | <i>lstPrevious()</i> | 2-357 |
| add new interfaces to internal | list. | <i>ripIfSearch()</i> | 2-638 |
| free tuples from linked | list. | <i>cisFree()</i> | 2-86 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|-----------------------------------|-------|
| add directory to access | list. | <i>tftpdDirectoryAdd()</i> | 2-900 |
| delete directory from access | list. | <i>tftpdDirectoryRemove()</i> | 2-900 |
| add node to end of | list. | <i>VXWList::add()</i> | 2-981 |
| report number of nodes in | list. | <i>VXWList::count()</i> | 2-982 |
| extract sublist from | list. | <i>VXWList::extract()</i> | 2-982 |
| find node in | list. | <i>VXWList::find()</i> | 2-982 |
| find first node in | list. | <i>VXWList::first()</i> | 2-983 |
| and return first node from | list. delete | <i>VXWList::get()</i> | 2-983 |
| find last node in | list. | <i>VXWList::last()</i> | 2-984 |
| find next node in | list. | <i>VXWList::next()</i> | 2-984 |
| find Nth node in | list. | <i>VXWList::nth()</i> | 2-985 |
| find previous node in | list. | <i>VXWList::previous()</i> | 2-986 |
| delete specified node from | list. | <i>VXWList::remove()</i> | 2-986 |
| initialize | list. | <i>VXWList::VXWList()</i> | 2-986 |
| free up | list. | <i>VXWList::~VXWList()</i> | 2-987 |
| insert node in | list after specified node. | <i>lstInsert()</i> | 2-355 |
| insert node in | list after specified node. | <i>VXWList::insert()</i> | 2-984 |
| | list all system-known devices. | <i>devs()</i> | 2-115 |
| initialize | list as copy of another. | <i>VXWList::VXWList()</i> | 2-987 |
| simple linked | list class. | <i>VXWList</i> | 1-417 |
| | list contents of directory. | <i>ls()</i> | 2-348 |
| directory. | list contents of RT-11 | <i>lsOld()</i> | 2-349 |
| initialize | list descriptor. | <i>lstInit()</i> | 2-354 |
| FTP. | list directory contents via | <i>ftpLs()</i> | 2-234 |
| from specified node. find | list node nStep steps away | <i>lstNStep()</i> | 2-356 |
| from specified node. find | list node nStep steps away | <i>VXWList::nStep()</i> | 2-985 |
| to SCSI controller. | list physical devices attached | <i>scsiShow()</i> | 2-695 |
| doubly linked | list subroutine library. | <i>lstLib</i> | 1-198 |
| | list symbols. | <i>lkup()</i> | 2-320 |
| near specified value. | list symbols whose values are | <i>lkAddr()</i> | 2-319 |
| concatenate two | lists. | <i>lstConcat()</i> | 2-350 |
| concatenate two | lists. | <i>VXWList::concat()</i> | 2-981 |
| initialize driver/ publish | ln network interface and | <i>lnattach()</i> | 2-325 |
| initialize driver and/ publish | lo network interface and | <i>loattach()</i> | 2-330 |
| instruction cache (i960). | load and lock I960Cx 1KB | <i>cacheI960CxIC1kLoadNLock()</i> | 2-58 |
| instruction cache (i960). | load and lock I960Cx 512-byte | <i>cacheI960CxICLoadNLock()</i> | 2-59 |
| instruction cache (i960). | load and lock I960Jx | <i>cacheI960JxICLoadNLock()</i> | 2-63 |
| | load driver into MUX. | <i>muxDevLoad()</i> | 2-473 |
| specified memory addresses. | load object module at | <i>VXWModule::VXWModule()</i> | 2-995 |
| memory. | load object module into | <i>ld()</i> | 2-313 |
| memory. | load object module into | <i>loadModule()</i> | 2-327 |
| memory. | load object module into | <i>loadModuleAt()</i> | 2-327 |
| memory. | load object module into | <i>VXWModule::VXWModule()</i> | 2-997 |
| object module | loader. | <i>loadLib</i> | 1-193 |
| device. issue | LOAD/UNLOAD command to SCSI | <i>scsiLoadUnit()</i> | 2-680 |
| convert bus address to | local address. | <i>sysBusToLocalAdrs()</i> | 2-845 |
| from Internet address. get | local address (host number) | <i>inet_lnaof()</i> | 2-269 |
| convert | local address to bus address. | <i>sysLocalToBusAdrs()</i> | 2-850 |
| address (VxMP Opt.). convert | local address to global | <i>smObjLocalToGlobal()</i> | 2-776 |
| convert global address to | local address (VxMP Opt.). | <i>smObjGlobalToLocal()</i> | 2-773 |

| | Keyword | Name | Page |
|--------------------------------|--|---------------------------------|-------|
| | initialize local debugging package. | <i>dbgInit()</i> | 2-110 |
| set appropriate | locale (ANSI). | <i>setLocale()</i> | 2-731 |
| ANSI | locale documentation. | <i>ansiLocale</i> | 1-13 |
| | lock access to shell. | <i>shellLock()</i> | 2-740 |
| cache. | lock all or part of specified | <i>cacheLock()</i> | 2-66 |
| into memory (POSIX). | lock all pages used by process | <i>mlockall()</i> | 2-433 |
| cache (i960). load and | lock I960Cx 1KB instruction | <i>cacheI960Cx1kLoadNLock()</i> | 2-58 |
| instruction cache/ load and | lock I960Cx 512-byte | <i>cacheI960CxICLoadNLock()</i> | 2-59 |
| (i960). load and | lock I960Jx instruction cache | <i>cacheI960JxICLoadNLock()</i> | 2-63 |
| | lock out interrupts. | <i>intLock()</i> | 2-280 |
| memory (POSIX). | lock specified pages into | <i>mlock()</i> | 2-432 |
| blocking if not available/ | lock (take) semaphore, | <i>sem_wait()</i> | 2-725 |
| returning error if/ | lock (take) semaphore, | <i>sem_trywait()</i> | 2-723 |
| SPARC,/ get current interrupt | lock-out level (MC680x0, | <i>intLockLevelGet()</i> | 2-282 |
| SPARC,/ set current interrupt | lock-out level (MC680x0, | <i>intLockLevelSet()</i> | 2-282 |
| | log formatted error message. | <i>logMsg()</i> | 2-344 |
| | log in to remote FTP server. | <i>ftpLogin()</i> | 2-234 |
| | log in to remote host. | <i>rlogin()</i> | 2-644 |
| | log out of VxWorks system. | <i>logout()</i> | 2-345 |
| compute base-2 | logarithm. | <i>log2()</i> | 2-334 |
| compute base-2 | logarithm. | <i>log2f()</i> | 2-335 |
| compute natural | logarithm (ANSI). | <i>log()</i> | 2-334 |
| compute base-10 | logarithm (ANSI). | <i>log10()</i> | 2-335 |
| compute base-10 | logarithm (ANSI). | <i>log10f()</i> | 2-336 |
| compute natural | logarithm (ANSI). | <i>logf()</i> | 2-336 |
| add | logging fd. | <i>logFdAdd()</i> | 2-337 |
| delete | logging fd. | <i>logFdDelete()</i> | 2-337 |
| set primary | logging fd. | <i>logFdSet()</i> | 2-338 |
| message | logging library. | <i>logLib</i> | 1-195 |
| initialize message | logging library. | <i>logInit()</i> | 2-340 |
| take spin-lock/ enable/disable | logging of failed attempts to | <i>smObjTimeoutLogEnable()</i> | 2-778 |
| VxWorks remote | login daemon. | <i>rlogind()</i> | 2-644 |
| initialize remote | login facility. | <i>rlogInit()</i> | 2-645 |
| remote | login library. | <i>rlogLib</i> | 1-308 |
| entry. display | login prompt and validate user | <i>loginPrompt()</i> | 2-341 |
| change | login string. | <i>loginStringSet()</i> | 2-341 |
| initialize | login table. | <i>loginInit()</i> | 2-340 |
| add user to | login table. | <i>loginUserAdd()</i> | 2-342 |
| delete user entry from | login table. | <i>loginUserDelete()</i> | 2-343 |
| display user | login table. | <i>loginUserShow()</i> | 2-343 |
| user name and password in | login table. verify | <i>loginUserVerify()</i> | 2-344 |
| print VxWorks | logo. | <i>printLogo()</i> | 2-595 |
| convert string to | long (ANSI). | <i>atol()</i> | 2-27 |
| compute absolute value of | long (ANSI). | <i>labs()</i> | 2-313 |
| notation Internet address to | long integer. convert dot | <i>inet_addr()</i> | 2-268 |
| convert string to | long integer (ANSI). | <i>strtol()</i> | 2-830 |
| convert string to unsigned | long integer (ANSI). | <i>strtoul()</i> | 2-831 |
| contents. do | long listing of directory | <i>ll()</i> | 2-320 |
| copy one buffer to another one | long word at a time. | <i>bcopyLongs()</i> | 2-30 |
| driver. software | loopback network interface | <i>if_loop</i> | 1-160 |

| | Keyword | Name | Page |
|--|--|--|--------|
| convert upper-case letter to lower-case equivalent (ANSI). | <code>tolower()</code> | <code>tolower()</code> | 2-921 |
| test whether character is lower-case letter (ANSI). | <code>islower()</code> | <code>islower()</code> | 2-307 |
| upper-case equivalent/ convert lower-case letter to | <code>toupper()</code> | <code>toupper()</code> | 2-922 |
| chip device driver for IBM-PC LPT. parallel | <code>lptDrv</code> | <code>lptDrv</code> | 1-197 |
| initialize LPT driver. | <code>lptDrv()</code> | <code>lptDrv()</code> | 2-347 |
| create device for LPT port. | <code>lptDevCreate()</code> | <code>lptDevCreate()</code> | 2-347 |
| show LPT statistics. | <code>lptShow()</code> | <code>lptShow()</code> | 2-348 |
| Interface: I/O DMA library/ LSI Logic L64862 MBus-to-SBus | <code>mmuL64862Lib</code> | <code>mmuL64862Lib</code> | 1-232 |
| driver. M68681 serial communications | <code>m68681Sio</code> | <code>m68681Sio</code> | 1-213 |
| initialize M68681_DUART. | <code>m68681DevInit()</code> | <code>m68681DevInit()</code> | 2-391 |
| initialize M68681_DUART, part 2. | <code>m68681DevInit2()</code> | <code>m68681DevInit2()</code> | 2-391 |
| initialize M68901_CHAN structure. | <code>m68901DevInit()</code> | <code>m68901DevInit()</code> | 2-395 |
| set signal mask. | <code>sigsetmask()</code> | <code>sigsetmask()</code> | 2-749 |
| address. extract net mask field from Internet | <code>bootNetmaskExtract()</code> | <code>bootNetmaskExtract()</code> | 2-38 |
| get subnet mask for network interface. | <code>ifMaskGet()</code> | <code>ifMaskGet()</code> | 2-263 |
| examine and/or change signal mask (POSIX). | <code>sigprocmask()</code> | <code>sigprocmask()</code> | 2-747 |
| contents of window invalid mask register (SPARC). return | <code>wim()</code> | <code>wim()</code> | 2-1064 |
| ANSI math documentation. | <code>ansiMath</code> | <code>ansiMath</code> | 1-13 |
| library to high-level math functions. C interface | <code>mathALib</code> | <code>mathALib</code> | 1-218 |
| hardware floating-point math library. | <code>mathHardLib</code> | <code>mathHardLib</code> | 1-220 |
| hardware floating-point math support. initialize | <code>mathHardInit()</code> | <code>mathHardInit()</code> | 2-402 |
| software floating-point math support. initialize | <code>mathSoftInit()</code> | <code>mathSoftInit()</code> | 2-402 |
| kernel/ enable MB86930 automatic locking of | <code>cacheMb930LockAuto()</code> | <code>cacheMb930LockAuto()</code> | 2-67 |
| clear line from MB86930 cache. | <code>cacheMb930ClearLine()</code> | <code>cacheMb930ClearLine()</code> | 2-66 |
| initialize Fujitsu MB86930 cache library. | <code>cacheMb930LibInit()</code> | <code>cacheMb930LibInit()</code> | 2-67 |
| management library. Fujitsu MB86930 (SPARC) cache | <code>cacheMb930Lib</code> | <code>cacheMb930Lib</code> | 1-46 |
| MB86940 UART tty driver. | <code>mb86940Sio</code> | <code>mb86940Sio</code> | 1-221 |
| interface/ END-style Fujitsu MB86960 Ethernet network | <code>mb86960End</code> | <code>mb86960End</code> | 1-221 |
| interface driver. Fujitsu MB86960 NICE Ethernet network | <code>if_fn</code> | <code>if_fn</code> | 1-151 |
| Controller (SPC)/ Fujitsu MB87030 SCSI Protocol | <code>mb87030Lib</code> | <code>mb87030Lib</code> | 1-223 |
| create control structure for MB87030 SPC. | <code>mb87030CtrlCreate()</code> | <code>mb87030CtrlCreate()</code> | 2-405 |
| control structure for MB87030 SPC. initialize | <code>mb87030CtrlInit()</code> | <code>mb87030CtrlInit()</code> | 2-406 |
| display values of all readable MB87030 SPC registers. | <code>mb87030Show()</code> | <code>mb87030Show()</code> | 2-407 |
| report mbuf statistics. | <code>mbufShow()</code> | <code>mbufShow()</code> | 2-414 |
| disable superscalar dispatch (MC68060). | <code>vxSSDisable()</code> | <code>vxSSDisable()</code> | 2-977 |
| enable superscalar dispatch (MC68060). | <code>vxSSEnable()</code> | <code>vxSSEnable()</code> | 2-977 |
| disable store buffer (MC68060 only). | <code>cacheStoreBufDisable()</code> | <code>cacheStoreBufDisable()</code> | 2-71 |
| enable store buffer (MC68060 only). | <code>cacheStoreBufEnable()</code> | <code>cacheStoreBufEnable()</code> | 2-71 |
| of register d0 (also d1 - d7) (MC680x0). return contents | <code>d0()</code> | <code>d0()</code> | 2-108 |
| of register a0 (also a1 - a7) (MC680x0). return contents | <code>a0()</code> | <code>a0()</code> | 2-1 |
| contents of status register (MC680x0). return | <code>sr()</code> | <code>sr()</code> | 2-806 |
| set task status register (MC680x0, MIPS, i386/i486). | <code>VXWTask::SRSet()</code> | <code>VXWTask::SRSet()</code> | 2-1037 |
| set task status register (MC680x0, MIPS, i386/i486). | <code>taskSRSet()</code> | <code>taskSRSet()</code> | 2-886 |
| ARM). set interrupt level (MC680x0, SPARC, i960, x86, | <code>intLevelSet()</code> | <code>intLevelSet()</code> | 2-279 |
| /interrupt lock-out level (MC680x0, SPARC, i960, x86,/ | <code>intLockLevelGet()</code> | <code>intLockLevelGet()</code> | 2-282 |
| /interrupt lock-out level (MC680x0, SPARC, i960, x86,/ | <code>intLockLevelSet()</code> | <code>intLockLevelSet()</code> | 2-282 |
| ARM). /exception vector table (MC680x0, SPARC, i960, x86, | <code>intVecTableWriteProtect()</code> | <code>intVecTableWriteProtect()</code> | 2-288 |
| MIPS). /handler for C routine (MC680x0, SPARC, i960, x86, | <code>intHandlerCreate()</code> | <code>intHandlerCreate()</code> | 2-279 |
| MIPS). get interrupt vector (MC680x0, SPARC, i960, x86, | <code>intVecGet()</code> | <code>intVecGet()</code> | 2-286 |
| MIPS). set CPU vector (trap) (MC680x0, SPARC, i960, x86, | <code>intVecSet()</code> | <code>intVecSet()</code> | 2-286 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|-----------------------------------|-------|
| get vector (trap) base address | (MC680x0, SPARC, i960, x86,/ | <i>intVecBaseGet()</i> | 2-284 |
| set vector (trap) base address | (MC680x0, SPARC, i960, x86,/ | <i>intVecBaseSet()</i> | 2-285 |
| Motorola | MC68302 bimodal tty driver. | m68302Sio | 1-210 |
| Motorola | MC68332 tty driver. | m68332Sio | 1-211 |
| driver. Motorola | MC68360 SCC UART serial | m68360Sio | 1-211 |
| | MC68562 DUSCC serial driver. | m68562Sio | 1-212 |
| | MC68901 MFP tty driver. | m68901Sio | 1-215 |
| system partition block of | memory (VxMP Opt.) /memory | <i>smMemFree()</i> | 2-761 |
| display | memory. | <i>d()</i> | 2-107 |
| load object module into | memory. | <i>ld()</i> | 2-313 |
| load object module into | memory. | <i>loadModule()</i> | 2-327 |
| load object module into | memory. | <i>loadModuleAt()</i> | 2-327 |
| modify | memory. | <i>m()</i> | 2-358 |
| allocate aligned | memory. | <i>memalign()</i> | 2-415 |
| processor write buffers to | memory. flush | <i>cachePipeFlush()</i> | 2-68 |
| free block of | memory. | <i>cfree()</i> | 2-83 |
| get address of top of logical | memory. | <i>sysMemTop()</i> | 2-851 |
| get address of top of | memory. | <i>sysPhysMemTop()</i> | 2-853 |
| load object module into | memory. | <i>VXWModule::VXWModule()</i> | 2-997 |
| object module at specified | memory addresses. load | <i>VXWModule::VXWModule()</i> | 2-995 |
| free block of | memory (ANSI). | <i>free()</i> | 2-220 |
| compare two blocks of | memory (ANSI). | <i>memcmp()</i> | 2-416 |
| set block of | memory (ANSI). | <i>memset()</i> | 2-428 |
| reallocate block of | memory (ANSI). | <i>realloc()</i> | 2-615 |
| interface driver. shared | memory backplane network | if_sm | 1-166 |
| create and initialize shared | memory binary semaphore (VxMP/ | <i>semBSmCreate()</i> | 2-708 |
| behavior common to all shared | memory classes. naming | VXWSmName | 1-425 |
| create and initialize shared | memory counting semaphore/ | <i>semCSmCreate()</i> | 2-710 |
| /default run-time support for | memory deallocation (C++). | <i>operator delete()</i> | 2-543 |
| create | memory device. | <i>memDevCreate()</i> | 2-417 |
| delete | memory device. | <i>memDevDelete()</i> | 2-419 |
| pseudo | memory device driver. | memDrv | 1-226 |
| files. create | memory device for multiple | <i>memDevCreateDir()</i> | 2-419 |
| create PCMCIA | memory disk device. | <i>sramDevCreate()</i> | 2-806 |
| install | memory driver. | <i>memDrv()</i> | 2-420 |
| install PCMCIA SRAM | memory driver. | <i>sramDrv()</i> | 2-807 |
| memory system/ allocate | memory for array from shared | <i>smMemCalloc()</i> | 2-760 |
| search block of | memory for character (ANSI). | <i>memchr()</i> | 2-416 |
| initialize | memory for chip. | <i>mb86960MemInit()</i> | 2-405 |
| initialize | memory for chip. | <i>mbcMemInit()</i> | 2-410 |
| initialize | memory for chip. | <i>ntMemInit()</i> | 2-538 |
| initialize | memory for chip. | <i>ultraMemInit()</i> | 2-938 |
| initialize | memory for Lance chip. | <i>ln7990InitMem()</i> | 2-323 |
| another (ANSI). copy | memory from one location to | <i>memcpy()</i> | 2-417 |
| another (ANSI). copy | memory from one location to | <i>memmove()</i> | 2-421 |
| allocate aligned | memory from partition. | <i>memPartAlignedAlloc()</i> | 2-422 |
| allocate block of | memory from partition. | <i>memPartAlloc()</i> | 2-423 |
| allocate aligned | memory from partition. | <i>VXWMemPart::alignedAlloc()</i> | 2-988 |
| allocate block of | memory from partition. | <i>VXWMemPart::alloc()</i> | 2-989 |
| partition/ reallocate block of | memory from shared mem system | <i>smMemRealloc()</i> | 2-763 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|---------------------------------|--------|
| system/ allocate block of | memory from shared memory | <i>smMemMalloc()</i> | 2-762 |
| partition/ allocate block of | memory from system memory | <i>malloc()</i> | 2-396 |
| free block of | memory in partition. | <i>memPartFree()</i> | 2-424 |
| free block of | memory in partition. | <i>VXWMemPart::free()</i> | 2-989 |
| reallocate block of | memory in partition. | <i>VXWMemPart::realloc()</i> | 2-991 |
| reallocate block of | memory in specified partition. | <i>memPartRealloc()</i> | 2-426 |
| (POSIX). | memory management library | <i>mmanPxLib</i> | 1-231 |
| (VxMP Opt.). shared | memory management library | <i>smMemLib</i> | 1-355 |
| routines (VxMP Opt.). shared | memory management show | <i>smMemShow</i> | 1-357 |
| (VxMP Opt.). shared | memory message queue library | <i>msgQSmLib</i> | 1-252 |
| create and initialize shared | memory message queue (VxMP/ | <i>msgQSmCreate()</i> | 2-466 |
| remove object from shared | memory name database | <i>VXWSmName::~VXWSmName()</i> | 2-1021 |
| Opt.). add name to shared | memory name database (VxMP | <i>smNameAdd()</i> | 2-765 |
| show information about shared | memory network. | <i>smNetShow()</i> | 2-772 |
| VxWorks interface to shared | memory network (backplane)/ | <i>smNetLib</i> | 1-360 |
| initialize shared | memory network driver. | <i>smNetInit()</i> | 2-771 |
| routines. shared | memory network driver show | <i>smNetShow</i> | 1-361 |
| attach shared | memory network interface. | <i>smNetAttach()</i> | 2-769 |
| address associated with shared | memory network interface. get | <i>smNetInetGet()</i> | 2-770 |
| Opt.). look up shared | memory object by name (VxMP | <i>smNameFind()</i> | 2-766 |
| Opt.). look up shared | memory object by value (VxMP | <i>smNameFindByValue()</i> | 2-767 |
| shared | memory object initialization. | <i>usrSmObj</i> | 1-413 |
| get name and type of shared | memory object (VxMP Opt.). | <i>VXWSmName::nameGet()</i> | 2-1019 |
| get name of shared | memory object (VxMP Opt.). | <i>VXWSmName::nameGet()</i> | 2-1019 |
| initialize shared | memory objects. | <i>usrSmObjInit()</i> | 2-951 |
| (VxMP/ initialize shared | memory objects descriptor | <i>smObjInit()</i> | 2-774 |
| attach calling CPU to shared | memory objects facility (VxMP/ | <i>smObjAttach()</i> | 2-772 |
| Opt.). install shared | memory objects facility (VxMP | <i>smObjLibInit()</i> | 2-775 |
| Opt.). initialize shared | memory objects facility (VxMP | <i>smObjSetup()</i> | 2-776 |
| Opt.). shared | memory objects library (VxMP | <i>smObjLib</i> | 1-361 |
| library (VxMP Opt.). shared | memory objects name database | <i>smNameLib</i> | 1-357 |
| show routines (VxMP/ shared | memory objects name database | <i>smNameShow</i> | 1-360 |
| remove object from shared | memory objects name database/ | <i>smNameRemove()</i> | 2-768 |
| (VxMP/ show contents of shared | memory objects name database | <i>smNameShow()</i> | 2-768 |
| (VxMP Opt.). shared | memory objects show routines | <i>smObjShow</i> | 1-364 |
| / current status of shared | memory objects (VxMP Opt.). | <i>smObjShow()</i> | 2-777 |
| allocate | memory on page boundary. | <i>valloc()</i> | 2-955 |
| address space. map PCMCIA | memory onto specified ISA | <i>sramMap()</i> | 2-807 |
| add memory to system | memory partition. | <i>memAddToPool()</i> | 2-415 |
| largest free block in system | memory partition. find | <i>memFindMax()</i> | 2-420 |
| set debug options for system | memory partition. | <i>memOptionsSet()</i> | 2-421 |
| add memory to | memory partition. | <i>memPartAddToPool()</i> | 2-422 |
| create | memory partition. | <i>memPartCreate()</i> | 2-423 |
| set debug options for | memory partition. | <i>memPartOptionsSet()</i> | 2-425 |
| add memory to | memory partition. | <i>VXWMemPart::addToPool()</i> | 2-988 |
| set debug options for | memory partition. | <i>VXWMemPart::options()</i> | 2-990 |
| create | memory partition. | <i>VXWMemPart::VXWMemPart()</i> | 2-992 |
| /block of memory from system | memory partition (ANSI). | <i>malloc()</i> | 2-396 |
| statistics. show system | memory partition blocks and | <i>memShow()</i> | 2-429 |
| | memory partition classes. | <i>VXWMemPart</i> | 1-419 |

| | Keyword | Name | Page |
|--|--------------------------------------|--------------------------------------|--------------------------------------|
| | full-featured | memory partition manager. | memLib 1-228 |
| | core | memory partition manager. | memPartLib 1-230 |
| | facility. initialize | memory partition show | memShowInit() 2-430 |
| | create shared | memory partition (VxMP Opt.). | memPartSmCreate() 2-427 |
| | construct back to | memory pool. /clBlk-cluster | netCIBlkFree() 2-504 |
| | free cluster back to | memory pool. | netCIFree() 2-505 |
| | free mBlk back to | memory pool. | netMblkFree() 2-513 |
| | delete | memory pool. | netPoolDelete() 2-515 |
| | initialize netBufLib -managed | memory pool. | netPoolInit() 2-515 |
| | lock specified pages into | memory (POSIX). | mlock() 2-432 |
| | all pages used by process into | memory (POSIX). lock | mlockall() 2-433 |
| | Opt.). shared | memory semaphore library (VxMP | semSmLib 1-347 |
| | | memory show routines. | memShow 1-231 |
| | of memory (VxMP/ free shared | memory system partition block | smMemFree() 2-761 |
| | and statistics/ show shared | memory system partition blocks | smMemShow() 2-764 |
| | Opt.). add memory to shared | memory system partition (VxMP | smMemAddToPool() 2-759 |
| | /memory for array from shared | memory system partition (VxMP/ | smMemMalloc() 2-760 |
| | /largest free block in shared | memory system partition (VxMP/ | smMemFindMax() 2-760 |
| | /block of memory from shared | memory system partition (VxMP/ | smMemMalloc() 2-762 |
| | set debug options for shared | memory system partition (VxMP/ | smMemOptionsSet() 2-762 |
| | add | memory to memory partition. | memPartAddToPool() 2-422 |
| | add | memory to memory partition. | VXWMemPart::addToPool() 2-988 |
| | partition (VxMP Opt.). add | memory to shared memory system | smMemAddToPool() 2-759 |
| | partition. add | memory to system memory | memAddToPool() 2-415 |
| | log formatted error | message. | logMsg() 2-344 |
| | completion of transmission | message. transmit | masterLpcComp() 2-398 |
| | send BOOTP request | message. | bootpMsgSend() 2-40 |
| | build identification | message. | scsiIdentMsgBuild() 2-678 |
| | parse identification | message. | scsiIdentMsgParse() 2-678 |
| | agent. build and encode | message and send it to master | saMsgBuild() 2-665 |
| | socket. create zbuf from user | message and send it to UDP | zbufSockBufSendto() 2-1079 |
| | error number in errno to error | message (ANSI). map | perror() 2-569 |
| | receive | message from message queue. | VXWMsgQ::receive() 2-1004 |
| | receive | message from message queue. | msgQReceive() 2-462 |
| | (POSIX). receive | message from message queue | mq_receive() 2-451 |
| | receive | message from socket. | recvfrom() 2-618 |
| | receive | message from socket. | recvmsg() 2-618 |
| | socket. receive | message in zbuf from UDP | zbufSockRecvfrom() 2-1081 |
| | (POSIX). notify task that | message is available on queue | mq_notify() 2-449 |
| | post-processing when outgoing | message is rejected. perform | scsiMsgOutReject() 2-686 |
| | post-processing after SCSI | message is sent. perform | scsiMsgOutComplete() 2-685 |
| | | message logging library. | logLib 1-195 |
| | initialize | message logging library. | logInit() 2-340 |
| | subagent. asynchronous | message processing routine for | smpSaHandlerAsync() 2-788 |
| | get information about | message queue. | VXWMsgQ::info() 2-1002 |
| | receive message from | message queue. | VXWMsgQ::receive() 2-1004 |
| | send message to | message queue. | VXWMsgQ::send() 2-1005 |
| | show information about | message queue. | VXWMsgQ::show() 2-1007 |
| | create and initialize | message queue. | VXWMsgQ::VXWMsgQ() 2-1008 |
| | delete | message queue. | VXWMsgQ::~VXWMsgQ() 2-1009 |

| | Keyword | Name | Page |
|-------------------------------|--------------------------------------|--------------------------------|--------|
| create and initialize | message queue. | <i>msgQCreate()</i> | 2-459 |
| delete | message queue. | <i>msgQDelete()</i> | 2-459 |
| get information about | message queue. | <i>msgQInfoGet()</i> | 2-460 |
| number of messages queued to | message queue. get | <i>msgQNumMsgs()</i> | 2-462 |
| receive message from | message queue. | <i>msgQReceive()</i> | 2-462 |
| send message to | message queue. | <i>msgQSend()</i> | 2-463 |
| show information about | message queue. | <i>msgQShow()</i> | 2-464 |
| send packet to master agent's | message queue. | <i>saloWrite()</i> | 2-664 |
| (POSIX). get | message queue attributes | <i>mq_getattr()</i> | 2-448 |
| (POSIX). set | message queue attributes | <i>mq_setattr()</i> | 2-453 |
| | message queue classes. | <i>VXWMsgQ</i> | 1-421 |
| | message queue library. | <i>msgQLib</i> | 1-251 |
| initialize POSIX | message queue library. | <i>mqPxLibInit()</i> | 2-447 |
| | message queue library (POSIX). | <i>mqPxLib</i> | 1-249 |
| Opt.). shared memory | message queue library (VxMP) | <i>msgQSmLib</i> | 1-252 |
| close | message queue (POSIX). | <i>mq_close()</i> | 2-448 |
| open | message queue (POSIX). | <i>mq_open()</i> | 2-450 |
| receive message from | message queue (POSIX). | <i>mq_receive()</i> | 2-451 |
| send message to | message queue (POSIX). | <i>mq_send()</i> | 2-452 |
| remove | message queue (POSIX). | <i>mq_unlink()</i> | 2-454 |
| POSIX | message queue show. | <i>mqPxShow</i> | 1-250 |
| initialize POSIX | message queue show facility. | <i>mqPxShowInit()</i> | 2-447 |
| initialize | message queue show facility. | <i>msgQShowInit()</i> | 2-465 |
| | message queue show routines. | <i>msgQShow</i> | 1-252 |
| /and initialize shared memory | message queue (VxMP Opt.). | <i>msgQSmCreate()</i> | 2-466 |
| handle complete SCSI | message received from target. | <i>scsiMsgInComplete()</i> | 2-685 |
| send | message to message queue. | <i>VXWMsgQ::send()</i> | 2-1005 |
| send | message to message queue. | <i>msgQSend()</i> | 2-463 |
| (POSIX). send | message to message queue | <i>mq_send()</i> | 2-452 |
| send TFTP | message to remote system. | <i>tftpSend()</i> | 2-906 |
| send | message to socket. | <i>sendmsg()</i> | 2-726 |
| send | message to socket. | <i>sendto()</i> | 2-727 |
| send | message to subagent. | <i>masterIpcSend()</i> | 2-401 |
| send zbuf | message to UDP socket. | <i>zbufSockSendto()</i> | 2-1083 |
| get IP | MIB-II address entry. | <i>m2IpAddrTblEntryGet()</i> | 2-363 |
| agents. | MIB-II API library for SNMP | <i>m2Lib</i> | 1-203 |
| add, modify, or delete | MIB-II ARP entry. | <i>m2IpAtransTblEntrySet()</i> | 2-365 |
| get | MIB-II ARP table entry. | <i>m2IpAtransTblEntryGet()</i> | 2-364 |
| listeners. get UDP | MIB-II entry from UDP list of | <i>m2UdpTblEntryGet()</i> | 2-384 |
| change | MIB-II error count. | <i>mib2ErrorAdd()</i> | 2-430 |
| initialize | MIB-II ICMP-group access. | <i>m2IcmpInit()</i> | 2-360 |
| Agents. | MIB-II ICMP-group API for SNMP | <i>m2IcmpLib</i> | 1-199 |
| variables. get | MIB-II ICMP-group global | <i>m2IcmpGroupInfoGet()</i> | 2-359 |
| or DOWN. set state of | MIB-II interface entry to UP | <i>m2IfTblEntrySet()</i> | 2-362 |
| SNMP agents. | MIB-II interface-group API for | <i>m2IfLib</i> | 1-200 |
| routines. initialize | MIB-II interface-group | <i>m2IfInit()</i> | 2-361 |
| variables. get | MIB-II interface-group scalar | <i>m2IfGroupInfoGet()</i> | 2-360 |
| entry. get | MIB-II interface-group table | <i>m2IfTblEntryGet()</i> | 2-361 |
| initialize | MIB-II IP-group access. | <i>m2IpInit()</i> | 2-367 |
| agents. | MIB-II IP-group API for SNMP | <i>m2IpLib</i> | 1-201 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|---------------------------------|--------|
| variables. get | MIB-II IP-group scalar | <i>m2IpGroupInfoGet()</i> | 2-366 |
| new values. set | MIB-II IP-group variables to | <i>m2IpGroupInfoSet()</i> | 2-366 |
| delete all | MIB-II library groups. | <i>m2Delete()</i> | 2-358 |
| set | MIB-II routing table entry. | <i>m2IpRouteTblEntrySet()</i> | 2-368 |
| initialize | MIB-II structure. | <i>mib2Init()</i> | 2-431 |
| resources used to access | MIB-II system group. delete | <i>m2SysDelete()</i> | 2-379 |
| SNMP agents. | MIB-II system-group API for | <i>m2SysLib</i> | 1-206 |
| initialize | MIB-II system-group routines. | <i>m2SysInit()</i> | 2-380 |
| entry. get | MIB-II TCP connection table | <i>m2TcpConnEntryGet()</i> | 2-381 |
| initialize | MIB-II TCP-group access. | <i>m2TcpInit()</i> | 2-383 |
| agents. | MIB-II TCP-group API for SNMP | <i>m2TcpLib</i> | 1-207 |
| variables. get | MIB-II TCP-group scalar | <i>m2TcpGroupInfoGet()</i> | 2-382 |
| initialize | MIB-II UDP-group access. | <i>m2UdpInit()</i> | 2-384 |
| agents. | MIB-II UDP-group API for SNMP | <i>m2UdpLib</i> | 1-209 |
| variables. get | MIB-II UDP-group scalar | <i>m2UdpGroupInfoGet()</i> | 2-384 |
| get system-group | MIB-II variables. | <i>m2SysGroupInfoGet()</i> | 2-379 |
| values. set system-group | MIB-II variables to new | <i>m2SysGroupInfoSet()</i> | 2-380 |
| initialize | microSPARC cache library. | <i>cacheMicroSparcLibInit()</i> | 2-68 |
| library. | microSPARC cache management | <i>cacheMicroSparcLib</i> | 1-46 |
| library. | microSparc I/II I/O DMA | <i>ioMmuMicroSparcLib</i> | 1-180 |
| structures. initialize | microSparc I/II I/O MMU data | <i>ioMmuMicroSparcInit()</i> | 2-292 |
| map I/O MMU for | microSparc I/II/ | <i>ioMmuMicroSparcMap()</i> | 2-292 |
| (MC680x0, SPARC, i960, x86, | MIPS). /handler for C routine | <i>intHandlerCreate()</i> | 2-279 |
| (MC680x0, SPARC, i960, x86, | MIPS). get interrupt vector | <i>intVecGet()</i> | 2-286 |
| (MC680x0, SPARC, i960, x86, | MIPS). set CPU vector (trap) | <i>intVecSet()</i> | 2-286 |
| (MC680x0, SPARC, i960, x86, | MIPS, ARM). /base address | <i>intVecBaseGet()</i> | 2-284 |
| (MC680x0, SPARC, i960, x86, | MIPS, ARM). /base address | <i>intVecBaseSet()</i> | 2-285 |
| task status register (MC680x0, | MIPS, i386/i486). set | <i>VXWTask::SRSet()</i> | 2-1037 |
| task status register (MC680x0, | MIPS, i386/i486). set | <i>taskSRSet()</i> | 2-886 |
| assembly routines. | MIPS R3000 cache management | <i>cacheR3kALib</i> | 1-47 |
| library. | MIPS R3000 cache management | <i>cacheR3kLib</i> | 1-47 |
| library. | MIPS R33000 cache management | <i>cacheR33kLib</i> | 1-48 |
| library. | MIPS R333x0 cache management | <i>cacheR333x0Lib</i> | 1-49 |
| library. | MIPS R4000 cache management | <i>cacheR4kLib</i> | 1-48 |
| initialize microSparc I/II I/O | MMU data structures. | <i>ioMmuMicroSparcInit()</i> | 2-292 |
| initialize L64862 I/O | MMU DMA data structures/ | <i>mmuL64862DmaInit()</i> | 2-433 |
| (TMS390S10/MB86904). map I/O | MMU for microSparc I/II | <i>ioMmuMicroSparcMap()</i> | 2-292 |
| initialize | MMU for ROM (SPARC). | <i>mmuSparcRomInit()</i> | 2-434 |
| ROM | MMU initialization (SPARC). | <i>mmuSparcILib</i> | 1-237 |
| | mmu library for PentiumPro/II. | <i>mmuPro32Lib</i> | 1-232 |
| return | model name of CPU board. | <i>sysModel()</i> | 2-852 |
| device. issue | MODE_SELECT command to SCSI | <i>scsiModeSelect()</i> | 2-684 |
| tape device. issue | MODE_SELECT command to SCSI | <i>scsiTapeModeSelect()</i> | 2-697 |
| device. issue | MODE_SENSE command to SCSI | <i>scsiModeSense()</i> | 2-684 |
| tape device. issue | MODE_SENSE command to SCSI | <i>scsiTapeModeSense()</i> | 2-698 |
| transfer control to ROM | monitor. | <i>sysToMonitor()</i> | 2-860 |
| interface driver. | Motorola 68302fads END network | <i>mbcEnd</i> | 1-223 |
| network-interface driver. | Motorola 68EN302 | <i>if_mbc</i> | 1-160 |
| interface driver. | Motorola CPM core network | <i>if_cpm</i> | 1-126 |
| interface driver. END style | Motorola FEC Ethernet network | <i>motFecEnd</i> | 1-242 |

| | Keyword | Name | Page |
|--------------------------------|--|-----------------------------------|--------|
| | driver. Motorola MC68302 bimodal tty | m68302Sio | 1-210 |
| | Motorola MC68332 tty driver. | m68332Sio | 1-211 |
| | serial driver. Motorola MC68360 SCC UART | m68360Sio | 1-211 |
| network interface/ END style | Motorola MC68EN360/MPC800 | motCpmEnd | 1-238 |
| serial driver. | Motorola MPC800 SMC UART | ppc860Sio | 1-292 |
| exported by specified host. | mount all file systems | nfsMountAll() | 2-531 |
| initialize | mount daemon. | mountdInit() | 2-446 |
| initialize device and | mount DOS file system. | pccardMkfs() | 2-552 |
| | mount DOS file system. | pccardMount() | 2-552 |
| hard disk. | mount DOS file system from ATA | usrAtaConfig() | 2-947 |
| floppy disk. | mount DOS file system from | usrFdConfig() | 2-948 |
| hard disk. | mount DOS file system from IDE | usrIdeConfig() | 2-949 |
| | mount NFS file system. | nfsMount() | 2-530 |
| | Mount protocol library. | mountLib | 1-248 |
| display | mounted NFS devices. | nfsDevShow() | 2-525 |
| Motorola | MPC800 SMC UART serial driver. | ppc860Sio | 1-292 |
| CL-CD2400 | MPCC serial driver. | cd2400Sio | 1-52 |
| Register). disable | MTRR (Memory Type Range | pentiumMtrrDisable() | 2-560 |
| Register). enable | MTRR (Memory Type Range | pentiumMtrrEnable() | 2-560 |
| get MTRRs to specified | MTRR table. | pentiumMtrrGet() | 2-560 |
| set MTRRs from specified | MTRR table with WRMSR/ | pentiumMtrrSet() | 2-561 |
| table with WRMSR/ set | MTRRs from specified MTRR | pentiumMtrrSet() | 2-561 |
| get | MTRRs to specified MTRR table. | pentiumMtrrGet() | 2-560 |
| character/ convert | multibyte character to wide | mbtowc() | 2-414 |
| convert wide character to | multibyte character/ | wctomb() | 2-1049 |
| calculate length of | multibyte character/ | mblen() | 2-413 |
| chars/ convert series of | multibyte chars to wide | mbstowcs() | 2-413 |
| series of wide chars to | multibyte chars/ convert | wcstombs() | 2-1049 |
| create and initialize | mutual-exclusion semaphore. | semMCreate() | 2-714 |
| create and initialize | mutual-exclusion semaphore. | VXWMSem::VXWMSem() | 2-999 |
| library. | mutual-exclusion semaphore | semMLib | 1-341 |
| without restrictions. give | mutual-exclusion semaphore | semMGiveForce() | 2-715 |
| without restrictions. give | mutual-exclusion semaphore | VXWMSem::giveForce() | 2-999 |
| /registers for | NCR 53C710. | ncr710SetHwRegisterScsi2() | 2-490 |
| (SIOP) library (SCSI-1). | NCR 53C710 SCSI I/O Processor | ncr710Lib | 1-255 |
| (SIOP) library (SCSI-2). | NCR 53C710 SCSI I/O Processor | ncr710Lib2 | 1-256 |
| create control structure for | NCR 53C710 SIOP. | ncr710CtrlCreate() | 2-485 |
| create control structure for | NCR 53C710 SIOP. | ncr710CtrlCreateScsi2() | 2-486 |
| control structure for | NCR 53C710 SIOP. initialize | ncr710CtrlInit() | 2-487 |
| control structure for | NCR 53C710 SIOP. initialize | ncr710CtrlInitScsi2() | 2-488 |
| /registers for | NCR 53C710 SIOP. | ncr710SetHwRegister() | 2-489 |
| display values of all readable | NCR 53C710 SIOP registers. | ncr710Show() | 2-491 |
| display values of all readable | NCR 53C710 SIOP registers. | ncr710ShowScsi2() | 2-492 |
| Processor (SIOP) library/ | NCR 53C8xx PCI SCSI I/O | ncr810Lib | 1-257 |
| create control structure for | NCR 53C8xx SIOP. | ncr810CtrlCreate() | 2-493 |
| control structure for | NCR 53C8xx SIOP. initialize | ncr810CtrlInit() | 2-494 |
| /registers for | NCR 53C8xx SIOP. | ncr810SetHwRegister() | 2-495 |
| display values of all readable | NCR 53C8xx SIOP registers. | ncr810Show() | 2-496 |
| Controller (ASC) library/ | NCR 53C90 Advanced SCSI | ncr5390Lib1 | 1-258 |
| Controller (ASC) library/ | NCR 53C90 Advanced SCSI | ncr5390Lib2 | 1-259 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------|---|--------|
| | create control structure for | NCR 53C90 ASC. <i>ncr5390CtrlCreate()</i> | 2-497 |
| | create control structure for | NCR 53C90 ASC. <i>ncr5390CtrlCreateScsi2()</i> | 2-498 |
| | display values of all readable | NCR5390 chip registers. <i>ncr5390Show()</i> | 2-500 |
| | Controller library (SBIC). | NCR5390 SCSI-Bus Interface <i>ncr5390Lib</i> | 1-258 |
| | driver. | NE2000 END network interface <i>ne2000End</i> | 1-259 |
| | display statistics for | NE2000 ene network interface. <i>eneShow()</i> | 2-168 |
| | driver. Novell/Eagle | NE2000 network interface <i>if_ene</i> | 1-146 |
| | driver. | NEC 765 floppy disk device <i>nec765Fd</i> | 1-261 |
| | address. extract | net mask field from Internet <i>bootNetmaskExtract()</i> | 2-38 |
| | agent. initialize | NETROM packet device for WDB <i>wdbNetromPktDevInit()</i> | 2-1055 |
| | agent. | NETROM packet driver for WDB <i>wdbNetromPktDrv</i> | 1-433 |
| | create proxy ARP | network. <i>proxyNetCreate()</i> | 2-596 |
| | delete proxy | network. <i>proxyNetDelete()</i> | 2-597 |
| | route to destination that is | network. add <i>routeNetAdd()</i> | 2-655 |
| | about shared memory | network. show information <i>smNetShow()</i> | 2-772 |
| | notation, store in/ convert | network address from dot <i>inet_aton()</i> | 2-269 |
| | notation. extract | network address in dot <i>inet_netof_string()</i> | 2-271 |
| | notation, store it in/ convert | network address to dot <i>inet_ntoa_b()</i> | 2-273 |
| | decimal notation. convert | network address to dotted <i>inet_ntoa()</i> | 2-272 |
| | form Internet address from | network and host numbers. <i>inet_makeaddr()</i> | 2-270 |
| | form Internet address from | network and host numbers. <i>inet_makeaddr_b()</i> | 2-270 |
| | /interface to shared memory | network (backplane) driver. <i>smNetLib</i> | 1-360 |
| | network buffer library. | network buffer library. <i>netBufLib</i> | 1-262 |
| parameters with/ obtain set of | network configuration | network configuration <i>dhcpcBind()</i> | 2-115 |
| shut down | network connection. | network connection. <i>shutdown()</i> | 2-742 |
| control to boot ROMs. reset | network devices and transfer | network devices and transfer <i>reboot()</i> | 2-616 |
| DP83932B SONIC Ethernet | network driver. /Semiconductor | network driver. /Semiconductor <i>if_sn</i> | 1-167 |
| initialize shared memory | network driver. | network driver. <i>smNetInit()</i> | 2-771 |
| shared memory | network driver show routines. | network driver show routines. <i>smNetShow</i> | 1-361 |
| all configured Enhanced | Network Drivers. | Network Drivers. <i>muxShow()</i> | 2-483 |
| driver. | Network File System (NFS) I/O | Network File System (NFS) I/O <i>nfsDrv</i> | 1-268 |
| library. | Network File System (NFS) | Network File System (NFS) <i>nfsLib</i> | 1-270 |
| server library. | Network File System (NFS) | Network File System (NFS) <i>nfsdLib</i> | 1-266 |
| initialize | network host table. | network host table. <i>hostTblInit()</i> | 2-251 |
| routines. | network information display | network information display <i>netShow</i> | 1-265 |
| shows statistics for cs | network interface. | network interface. <i>csShow()</i> | 2-106 |
| publish dc | network interface. | network interface. <i>dcattach()</i> | 2-110 |
| statistics for SMC 8013WC elc | network interface. display | network interface. display <i>elcShow()</i> | 2-161 |
| statistics for 3C509 elt | network interface. display | network interface. display <i>eltShow()</i> | 2-163 |
| statistics for NE2000 ene | network interface. display | network interface. display <i>eneShow()</i> | 2-168 |
| display statistics for esmc | network interface. | network interface. <i>esmcShow()</i> | 2-175 |
| publish fei | network interface. | network interface. <i>feiattach()</i> | 2-197 |
| Add interface address for | network interface. | network interface. <i>ifAddrAdd()</i> | 2-258 |
| get Internet address of | network interface. | network interface. <i>ifAddrGet()</i> | 2-258 |
| set interface address for | network interface. | network interface. <i>ifAddrSet()</i> | 2-259 |
| get broadcast address for | network interface. | network interface. <i>ifBroadcastGet()</i> | 2-259 |
| set broadcast address for | network interface. | network interface. <i>ifBroadcastSet()</i> | 2-260 |
| specify flags for | network interface. | network interface. <i>ifFlagSet()</i> | 2-262 |
| get subnet mask for | network interface. | network interface. <i>ifMaskGet()</i> | 2-263 |
| define subnet for | network interface. | network interface. <i>ifMaskSet()</i> | 2-264 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|-------------------------|-------|
| get metric for | network interface. | <i>ifMetricGet()</i> | 2-264 |
| delete routes associated with | network interface. | <i>ifRouteDelete()</i> | 2-265 |
| send packet on | network interface. | <i>muxPollSend()</i> | 2-482 |
| send packet out on | network interface. | <i>muxSend()</i> | 2-482 |
| delete PPP | network interface. | <i>pppDelete()</i> | 2-577 |
| initialize PPP | network interface. | <i>pppInit()</i> | 2-580 |
| attach shared memory | network interface. | <i>smNetAttach()</i> | 2-769 |
| associated with shared memory | network interface. /address | <i>smNetInetGet()</i> | 2-770 |
| display statistics for ultra | network interface. | <i>ultraShow()</i> | 2-940 |
| initialize/ publish cpm | network interface and | <i>cpmattach()</i> | 2-101 |
| initialize/ publish cs | network interface and | <i>csAttach()</i> | 2-105 |
| initialize/ publish esmc | network interface and | <i>esmcattach()</i> | 2-174 |
| initialize/ publish mbc | network interface and | <i>mbcattach()</i> | 2-408 |
| initialize driver/ publish eex | network interface and | <i>eexattach()</i> | 2-151 |
| initialize driver/ publish ei | network interface and | <i>eiattach()</i> | 2-154 |
| initialize driver/ publish ei | network interface and | <i>eihkattach()</i> | 2-155 |
| initialize driver/ publish elc | network interface and | <i>elcattach()</i> | 2-160 |
| initialize driver/ publish ene | network interface and | <i>eneattach()</i> | 2-167 |
| initialize driver/ publish fn | network interface and | <i>fnattach()</i> | 2-207 |
| initialize/ publish lnPci | network interface and | <i>lnPciattach()</i> | 2-326 |
| initialize driver/ publish sl | network interface and | <i>slattach()</i> | 2-756 |
| initialize driver/ publish sn | network interface and | <i>snattach()</i> | 2-779 |
| initialize driver/ publish lo | network interface and | <i>loattach()</i> | 2-330 |
| initialize driver/ publish ln | network interface and | <i>lnattach()</i> | 2-325 |
| lease request. assign | network interface and setup | <i>dhcpcInit()</i> | 2-122 |
| output packet to | network interface device. | <i>cpmStartOutput()</i> | 2-102 |
| output packet to | network interface device. | <i>mbcStartOutput()</i> | 2-412 |
| END style Intel 82557 Ethernet | network interface driver. | <i>fei82557End</i> | 1-114 |
| Motorola CPM core | network interface driver. | <i>if_cpm</i> | 1-126 |
| Crystal Semiconductor CS8900 | network interface driver. | <i>if_cs</i> | 1-129 |
| DEC 21x4x Ethernet LAN | network interface driver. | <i>if_dc</i> | 1-133 |
| Intel EtherExpress 16 | network interface driver. | <i>if_eex</i> | 1-136 |
| Intel 82596 Ethernet | network interface driver. | <i>if_ei</i> | 1-137 |
| SMC 8013WC Ethernet | network interface driver. | <i>if_elc</i> | 1-144 |
| 3Com 3C509 Ethernet | network interface driver. | <i>if_elt</i> | 1-144 |
| Novell/Eagle NE2000 | network interface driver. | <i>if_ene</i> | 1-146 |
| /Ethernet2 SMC-91c9x Ethernet | network interface driver. | <i>if_esmc</i> | 1-147 |
| Intel 82557 Ethernet | network interface driver. | <i>if_fei</i> | 1-149 |
| Fujitsu MB86960 NICE Ethernet | network interface driver. | <i>if_fn</i> | 1-151 |
| AMD Am7990 LANCE Ethernet | network interface driver. | <i>if_ln</i> | 1-153 |
| Am79C970 PCnet-PCI Ethernet | network interface driver. AMD | <i>if_lnPci</i> | 1-156 |
| software loopback | network interface driver. | <i>if_loop</i> | 1-160 |
| /Semiconductor ST-NIC Chip | network interface driver. | <i>if_nicEvb</i> | 1-163 |
| Serial Line IP (SLIP) | network interface driver. | <i>if_sl</i> | 1-164 |
| shared memory backplane | network interface driver. | <i>if_sm</i> | 1-166 |
| SMC Elite Ultra Ethernet | network interface driver. | <i>if_ultra</i> | 1-171 |
| END style Intel Olicom PCMCIA | network interface driver. | <i>iOlicomEnd</i> | 1-177 |
| style AMD 7990 LANCE Ethernet | network interface driver. END | <i>ln7990End</i> | 1-190 |
| /Fujitsu MB86960 Ethernet | network interface driver. | <i>mb86960End</i> | 1-221 |
| Motorola 68302fads END | network interface driver. | <i>mbcEnd</i> | 1-223 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|--------------------------------|-------|
| /Motorola MC68EN360/MPC800 | network interface driver. | motCpmEnd | 1-238 |
| style Motorola FEC Ethernet | network interface driver. END | motFecEnd | 1-242 |
| NE2000 END | network interface driver. | ne2000End | 1-259 |
| /Semiconductor ST-NIC Chip | network interface driver. | nicEvbEnd | 1-271 |
| SMC Ultra Elite END | network interface driver. | ultraEnd | 1-404 |
| style DEC 21x4x PCI Ethernet | network interface driver. END | dec21x4xEnd | 1-63 |
| /DEC 21x40 PCI Ethernet | network interface driver. | dec21x40End | 1-67 |
| END style Intel 82596 Ethernet | network interface driver. | ei82596End | 1-96 |
| publish and initialize nicEvb | network interface driver. | nicEvbattach() | 2-533 |
| 3COM 3C509. END | network interface driver for | elt3c509End | 1-104 |
| 3COM 3C90xB XL. END | network interface driver for | el3c90xEnd | 1-99 |
| hkv3500. Intel 82596 Ethernet | network interface driver for | if_eihk | 1-140 |
| User Level IP (VxSim). | network interface driver for | if_ulip | 1-170 |
| ULIP for vxSim for/ END | network interface driver to | ntEnd | 1-274 |
| change | network interface flags. | ifFlagChange() | 2-261 |
| get | network interface flags. | ifFlagGet() | 2-262 |
| specify | network interface hop count. | ifMetricSet() | 2-265 |
| routine. delete | network interface input hook | etherInputHookDelete() | 2-177 |
| handler. | network interface interrupt | mbcIntr() | 2-410 |
| | network interface library. | ifLib | 1-125 |
| MUX | network interface library. | muxLib | 1-254 |
| | network interface library. | netLib | 1-265 |
| routine. delete | network interface output hook | etherOutputHookDelete() | 2-181 |
| display attached | network interfaces. | ifShow() | 2-266 |
| /ULIP interface to list of | network interfaces (VxSim). | ulattach() | 2-934 |
| address. return | network number from Internet | inet_netof() | 2-271 |
| address. convert Internet | network number from string to | inet_network() | 2-272 |
| initialize | network package. | netLibInit() | 2-509 |
| install | network remote file driver. | netDrv() | 2-507 |
| driver. | network remote file I/O | netDrv | 1-263 |
| library. | network route manipulation | routeLib | 1-309 |
| print synopsis of | network routines. | netHelp() | 2-508 |
| display host and | network routing tables. | routeShow() | 2-656 |
| initialize | network show routines. | netShowInit() | 2-519 |
| attach routine for TCP/IP | network stack. generic | ipAttach() | 2-299 |
| detach routine for TCP/IP | network stack. generic | ipDetach() | 2-299 |
| statistics. show | network stack data pool | netStackDataPoolShow() | 2-519 |
| statistics. show | network stack system pool | netStackSysPoolShow() | 2-520 |
| | network task entry point. | netTask() | 2-520 |
| client library. Simple | Network Time Protocol (SNTP) | sntpLib | 1-366 |
| server library. Simple | Network Time Protocol (SNTP) | sntpsLib | 1-367 |
| time. initialize | network with DHCP at boot | dhcpcBootBind() | 2-116 |
| information from requested | NFS device. /configuration | nfsDevInfoGet() | 2-524 |
| unmount | NFS device. | nfsUnmount() | 2-532 |
| display mounted | NFS devices. | nfsDevShow() | 2-525 |
| create list of all | NFS devices in system. | nfsDevListGet() | 2-524 |
| install | NFS driver. | nfsDrv() | 2-526 |
| IO system driver number for | nfs driver. return | nfsDrvNumGet() | 2-526 |
| specify file system to be | NFS exported. | nfsExport() | 2-528 |
| mount | NFS file system. | nfsMount() | 2-530 |

| | Keyword | Name | Page |
|--|---|--------------------------------|-------|
| | display NFS help menu. | <i>nfsHelp()</i> | 2-529 |
| | initialize NFS server. | <i>nfsdInit()</i> | 2-525 |
| | get status of NFS server. | <i>nfsdStatusGet()</i> | 2-527 |
| | show status of NFS server. | <i>nfsdStatusShow()</i> | 2-527 |
| | parameters. get NFS UNIX authentication | <i>nfsAuthUnixGet()</i> | 2-522 |
| | parameters. modify NFS UNIX authentication | <i>nfsAuthUnixPrompt()</i> | 2-522 |
| | parameters. set NFS UNIX authentication | <i>nfsAuthUnixSet()</i> | 2-523 |
| | parameters. display NFS UNIX authentication | <i>nfsAuthUnixShow()</i> | 2-523 |
| | parameters. set ID number of NFS UNIX authentication | <i>nfsIdSet()</i> | 2-530 |
| | interface/ Fujitsu MB86960 NICE Ethernet network | <i>if_fn</i> | 1-151 |
| | get contents of non-volatile RAM. | <i>sysNvRamGet()</i> | 2-852 |
| | write to non-volatile RAM. | <i>sysNvRamSet()</i> | 2-853 |
| | initialize NS 16550 chip. | <i>evbNs16550HrdInit()</i> | 2-182 |
| | /interrupt for NS 16550 chip. | <i>evbNs16550Int()</i> | 2-183 |
| | NS 16550 UART tty driver. | <i>ns16550Sio</i> | 1-273 |
| | intialize NS16550 channel. | <i>ns16550DevInit()</i> | 2-535 |
| | PPC403GA evaluation. NS16550 serial driver for IBM | <i>evbNs16550Sio</i> | 1-111 |
| | get information about object module. | <i>moduleInfoGet()</i> | 2-441 |
| | reload object module. | <i>reld()</i> | 2-619 |
| | get information about object module. | <i>VXWModule::info()</i> | 2-993 |
| | create and initialize object module. | <i>VXWModule::VXWModule()</i> | 2-998 |
| | unload object module. | <i>VXWModule::~VXWModule()</i> | 2-998 |
| | memory addresses. load object module at specified | <i>VXWModule::VXWModule()</i> | 2-995 |
| | file name or module/ unload object module by specifying | <i>unld()</i> | 2-944 |
| | group number. unload object module by specifying | <i>unldByGroup()</i> | 2-945 |
| | module ID. unload object module by specifying | <i>unldByModuleId()</i> | 2-945 |
| | name and path. unload object module by specifying | <i>unldByNameAndPath()</i> | 2-946 |
| | object module class. | <i>VXWModule</i> | 1-420 |
| | load object module into memory. | <i>ld()</i> | 2-313 |
| | load object module into memory. | <i>loadModule()</i> | 2-327 |
| | load object module into memory. | <i>loadModuleAt()</i> | 2-327 |
| | load object module into memory. | <i>VXWModule::VXWModule()</i> | 2-997 |
| | object module loader. | <i>loadLib</i> | 1-193 |
| | library. object module management | <i>moduleLib</i> | 1-237 |
| | library. object module unloading | <i>unldLib</i> | 1-408 |
| | drive. get offset to first partition of | <i>usrAtaPartition()</i> | 2-947 |
| | (POSIX). open directory for searching | <i>opendir()</i> | 2-543 |
| | driver-specific/ validate open fd and return | <i>iosFdValue()</i> | 2-296 |
| | open file. | <i>open()</i> | 2-542 |
| | (POSIX). open file specified by fd | <i>fdopen()</i> | 2-195 |
| | (ANSI). open file specified by name | <i>fopen()</i> | 2-207 |
| | (ANSI). open file specified by name | <i>freopen()</i> | 2-221 |
| | open message queue (POSIX). | <i>mq_open()</i> | 2-450 |
| | open socket. | <i>socket()</i> | 2-799 |
| | port bound to it. open socket with privileged | <i>rresvport()</i> | 2-658 |
| | invert order of bytes in buffer. | <i>binvert()</i> | 2-35 |
| | up resources after query times out. free | <i>snmpMasterCleanup()</i> | 2-780 |
| | zero out buffer. | <i>bzero()</i> | 2-46 |
| | time (SPARC). zero out buffer eight bytes at a | <i>bzeroDoubles()</i> | 2-47 |
| | lock out interrupts. | <i>intLock()</i> | 2-280 |

| Keyword | Name | Page |
|--|--------------------------------|--------|
| log out of VxWorks system. | <i>logout()</i> | 2-345 |
| send packet out on network interface. | <i>muxSend()</i> | 2-482 |
| copy in (or stdin) to out (or stdout). | <i>copy()</i> | 2-93 |
| form Ethernet address into packet. | <i>endEtherAddressForm()</i> | 2-164 |
| locate addresses in packet. | <i>endEtherPacketAddrGet()</i> | 2-165 |
| get type from ethernet packet. | <i>etherTypeGet()</i> | 2-182 |
| form address into packet. | <i>muxAddressForm()</i> | 2-468 |
| addressing information from packet. get | <i>muxPacketAddrGet()</i> | 2-479 |
| return data from packet. | <i>muxPacketDataGet()</i> | 2-480 |
| compress DNS name in DNS packet. | <i>resolvDNComp()</i> | 2-623 |
| DNS compressed name from DNS packet. expand | <i>resolvDNExpand()</i> | 2-624 |
| return beginning of packet data. | <i>endEtherPacketDataGet()</i> | 2-165 |
| initialize pipe packet device. | <i>wdbPipePktDevInit()</i> | 2-1056 |
| initialize NETROM packet device for WDB agent. | <i>wdbNetromPktDevInit()</i> | 2-1055 |
| initialize SLIP packet device for WDB agent. | <i>wdbSlipPktDevInit()</i> | 2-1056 |
| UDP/IP. END based packet driver for lightweight | <i>wdbEndPktDrv</i> | 1-432 |
| UDP/IP. pipe packet driver for lightweight | <i>wdbPipePktDrv</i> | 1-433 |
| NETROM packet driver for WDB agent. | <i>wdbNetromPktDrv</i> | 1-433 |
| completion. encode packet for subagent IO | <i>snmpSaHandlerFinish()</i> | 2-791 |
| master agent or/ encode packet for transmission to | <i>snmpSubEncode()</i> | 2-794 |
| poll for packet from device driver. | <i>muxPollReceive()</i> | 2-481 |
| library. Packet InterNet Grouper (PING) | <i>pingLib</i> | 1-288 |
| send packet on Ethernet interface. | <i>etherOutput()</i> | 2-179 |
| send packet on network interface. | <i>muxPollSend()</i> | 2-482 |
| interface. send packet out on network | <i>muxSend()</i> | 2-482 |
| copy packet to interface. | <i>elcPut()</i> | 2-160 |
| copy packet to interface. | <i>enePut()</i> | 2-168 |
| copy packet to interface. | <i>esmcPut()</i> | 2-174 |
| copy packet to interface. | <i>ultraPut()</i> | 2-939 |
| message queue. send packet to master agent's | <i>saIoWrite()</i> | 2-664 |
| device. output packet to network interface | <i>cpmStartOutput()</i> | 2-102 |
| device. output packet to network interface | <i>mbcStartOutput()</i> | 2-412 |
| to receive all Ethernet input packets. add routine | <i>etherInputHookAdd()</i> | 2-176 |
| to receive all Ethernet output packets. add routine | <i>etherOutputHookAdd()</i> | 2-180 |
| receive all internet protocol packets. add routine to | <i>ipFilterHookAdd()</i> | 2-300 |
| push packets onto "interface". | <i>ulStartOutput()</i> | 2-936 |
| get architecture-dependent page block size (VxVMI Opt.). | <i>vmPageBlockSizeGet()</i> | 2-967 |
| allocate memory on page boundary. | <i>valloc()</i> | 2-955 |
| clear page from CY7C604 cache. | <i>cacheCy604ClearPage()</i> | 2-51 |
| clear page from Sun-4 cache. | <i>cacheSun4ClearPage()</i> | 2-73 |
| Opt.). get state of page of virtual memory (VxVMI | <i>vmStateGet()</i> | 2-968 |
| return page size. | <i>vmBasePageSizeGet()</i> | 2-958 |
| return page size (VxVMI Opt.). | <i>vmPageSizeGet()</i> | 2-967 |
| add memory to system memory partition. | <i>memAddToPool()</i> | 2-415 |
| free block in system memory partition. find largest | <i>memFindMax()</i> | 2-420 |
| options for system memory partition. set debug | <i>memOptionsSet()</i> | 2-421 |
| add memory to memory partition. | <i>memPartAddToPool()</i> | 2-422 |
| allocate aligned memory from partition. | <i>memPartAlignedAlloc()</i> | 2-422 |
| allocate block of memory from partition. | <i>memPartAlloc()</i> | 2-423 |
| create memory partition. | <i>memPartCreate()</i> | 2-423 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|-----------------------------------|-------|
| free block of memory in | partition. | <i>memPartFree()</i> | 2-424 |
| set debug options for memory | partition. | <i>memPartOptionsSet()</i> | 2-425 |
| block of memory in specified | partition. reallocate | <i>memPartRealloc()</i> | 2-426 |
| fields in SCSI logical | partition. initialize | <i>scsiBlkDevInit()</i> | 2-673 |
| add memory to memory | partition. | <i>VXWMemPart::addToPool()</i> | 2-988 |
| allocate aligned memory from | partition. | <i>VXWMemPart::alignedAlloc()</i> | 2-988 |
| allocate block of memory from | partition. | <i>VXWMemPart::alloc()</i> | 2-989 |
| free block of memory in | partition. | <i>VXWMemPart::free()</i> | 2-989 |
| set debug options for memory | partition. | <i>VXWMemPart::options()</i> | 2-990 |
| reallocate block of memory in | partition. | <i>VXWMemPart::realloc()</i> | 2-991 |
| create memory | partition. | <i>VXWMemPart::VXWMemPart()</i> | 2-992 |
| of memory from system memory | partition (ANSI). /block | <i>malloc()</i> | 2-396 |
| free shared memory system | partition block of memory/ | <i>smMemFree()</i> | 2-761 |
| statistics. show | partition blocks and | <i>memPartShow()</i> | 2-427 |
| show system memory | partition blocks and/ | <i>memShow()</i> | 2-429 |
| statistics. show | partition blocks and | <i>VXWMemPart::show()</i> | 2-992 |
| show shared memory system | partition blocks and/ | <i>smMemShow()</i> | 2-764 |
| memory | partition classes. | <i>VXWMemPart</i> | 1-419 |
| get | partition information. | <i>memPartInfoGet()</i> | 2-425 |
| get | partition information. | <i>VXWMemPart::info()</i> | 2-990 |
| full-featured memory | partition manager. | <i>memLib</i> | 1-228 |
| core memory | partition manager. | <i>memPartLib</i> | 1-230 |
| get offset to first | partition of drive. | <i>usrAtaPartition()</i> | 2-947 |
| block device. define logical | partition on SCSI | <i>scsiBlkDevCreate()</i> | 2-673 |
| initialize memory | partition show facility. | <i>memShowInit()</i> | 2-430 |
| create shared memory | partition (VxMP Opt.). | <i>memPartSmCreate()</i> | 2-427 |
| memory to shared memory system | partition (VxMP Opt.). add | <i>smMemAddToPool()</i> | 2-759 |
| from shared memory system | partition (VxMP Opt.). /array | <i>smMemCalloc()</i> | 2-760 |
| block in shared memory system | partition (VxMP Opt.). /free | <i>smMemFindMax()</i> | 2-760 |
| from shared memory system | partition (VxMP Opt.). /memory | <i>smMemMalloc()</i> | 2-762 |
| /for shared memory system | partition (VxMP Opt.). | <i>smMemOptionsSet()</i> | 2-762 |
| memory from shared mem system | partition (VxMP Opt.). /of | <i>smMemRealloc()</i> | 2-763 |
| associate device with | passFs file system functions. | <i>passFsDevInit()</i> | 2-549 |
| prepare to use | passFs library. | <i>passFsInit()</i> | 2-549 |
| system library (VxSim). | pass-through (to UNIX) file | <i>passFsLib</i> | 1-279 |
| file system library. | pass-through (to Windows NT) | <i>ntPassFsLib</i> | 1-274 |
| set remote user name and | password. | <i>iam()</i> | 2-254 |
| get current user name and | password. | <i>remCurIdGet()</i> | 2-620 |
| set remote user name and | password. | <i>remCurIdSet()</i> | 2-620 |
| default | password encryption routine. | <i>loginDefaultEncrypt()</i> | 2-338 |
| verify user name and | password in login table. | <i>loginUserVerify()</i> | 2-344 |
| get current default | path. | <i>getwd()</i> | 2-244 |
| get current default | path. | <i>ioDefPathGet()</i> | 2-289 |
| set current default | path. | <i>ioDefPathSet()</i> | 2-289 |
| find module by file name and | path. | <i>moduleFindByNameAndPath()</i> | 2-439 |
| set current default | path. | <i>chdir()</i> | 2-84 |
| by specifying name and | path. unload object module | <i>unldByNameAndPath()</i> | 2-946 |
| get current default | path (POSIX). | <i>getcwd()</i> | 2-240 |
| | PC CARD enabler library. | <i>pccardLib</i> | 1-280 |
| get information from | PC card's CIS. | <i>cisGet()</i> | 2-86 |

| | Keyword | Name | Page |
|--|-------------------------------|--|--------|
| | perform | PCI bus scan. <i>aic7880GetNumOfBuses()</i> | 2-6 |
| | read from | PCI config space. <i>aic7880ReadConfig()</i> | 2-6 |
| | read to | PCI config space. <i>aic7880WriteConfig()</i> | 2-7 |
| | driver. END style DEC 21x4x | PCI Ethernet network interface <i>dec21x4xEnd</i> | 1-63 |
| | driver. END-style DEC 21x40 | PCI Ethernet network interface <i>dec21x40End</i> | 1-67 |
| | library (SCSI-2). NCR 53C8xx | PCI SCSI I/O Processor (SIOP) <i>ncr810Lib</i> | 1-257 |
| | initialize | PCIC chip. <i>pcicInit()</i> | 2-554 |
| | show all configurations of | PCIC chip. <i>pcicShow()</i> | 2-554 |
| | show all configurations of | PCMCIA chip. <i>pcmciaShow()</i> | 2-555 |
| | | PCMCIA CIS library. <i>cisLib</i> | 1-56 |
| | | PCMCIA CIS show library. <i>cisShow</i> | 1-57 |
| | get | PCMCIA configuration register. <i>cisConfigregGet()</i> | 2-85 |
| | set | PCMCIA configuration register. <i>cisConfigregSet()</i> | 2-85 |
| | ATA/IDE (LOCAL and | PCMCIA) disk device driver. <i>ataDrv</i> | 1-24 |
| | show/ ATA/IDE (LOCAL and | PCMCIA) disk device driver <i>ataShow</i> | 1-27 |
| | all show routines for | PCMCIA drivers. initialize <i>pcmciaShowInit()</i> | 2-556 |
| | enable | PCMCIA Etherlink III card. <i>pccardEltEnabler()</i> | 2-551 |
| | facilities. generic | PCMCIA event-handling <i>pcmciaLib</i> | 1-281 |
| | initialize | PCMCIA event-handling package. <i>pcmciaInit()</i> | 2-555 |
| | handle task-level | PCMCIA events. <i>pcmciaid()</i> | 2-555 |
| | driver. Databook TCIC/2 | PCMCIA host bus adaptor chip <i>tcic</i> | 1-390 |
| | library. Intel 82365SL | PCMCIA host bus adaptor chip <i>pcic</i> | 1-280 |
| | show library. Intel 82365SL | PCMCIA host bus adaptor chip <i>pcicShow</i> | 1-281 |
| | show library. Databook TCIC/2 | PCMCIA host bus adaptor chip <i>tcicShow</i> | 1-391 |
| | create | PCMCIA memory disk device. <i>sramDevCreate()</i> | 2-806 |
| | ISA address space. map | PCMCIA memory onto specified <i>sramMap()</i> | 2-807 |
| | END style Intel Olicom | PCMCIA network interface/ <i>iOlicomEnd</i> | 1-177 |
| | | PCMCIA show library. <i>pcmciaShow</i> | 1-282 |
| | | PCMCIA SRAM device driver. <i>sramDrv</i> | 1-370 |
| | install | PCMCIA SRAM memory driver. <i>sramDrv()</i> | 2-807 |
| | return contents of | pcw register (i960). <i>pcw()</i> | 2-556 |
| | get name of connected | peer. <i>getpeername()</i> | 2-241 |
| | address of point-to-point | peer. get Internet <i>ifDstAddrGet()</i> | 2-260 |
| | library. | Pentium and PentiumPro <i>pentiumLib</i> | 1-286 |
| | specific routines. | Pentium and PentiumPro <i>pentiumALib</i> | 1-282 |
| | specific show routines. | Pentium and PentiumPro <i>pentiumShow</i> | 1-288 |
| | Pentium and | PentiumPro library. <i>pentiumLib</i> | 1-286 |
| | Pentium and | PentiumPro specific routines. <i>pentiumALib</i> | 1-282 |
| | routines. Pentium and | PentiumPro specific show <i>pentiumShow</i> | 1-288 |
| | reports. begin | periodic task activity <i>spy()</i> | 2-801 |
| | reports. run | periodic task activity <i>spyTask()</i> | 2-804 |
| | spawn task to call function | periodically. <i>period()</i> | 2-567 |
| | call function | periodically. <i>periodRun()</i> | 2-568 |
| | Packet InterNet Grouper | (PING) library. <i>pingLib</i> | 1-288 |
| | create | pipe device. <i>pipeDevCreate()</i> | 2-571 |
| | initialize | pipe driver. <i>pipeDrv()</i> | 2-572 |
| | | pipe I/O driver. <i>pipeDrv</i> | 1-289 |
| | initialize | pipe packet device. <i>wdbPipePktDevInit()</i> | 2-1056 |
| | lightweight UDP/IP. | pipe packet driver for <i>wdbPipePktDrv</i> | 1-433 |
| | address for other end of | point-to-point link. define <i>ifDstAddrSet()</i> | 2-261 |

| | Keyword | Name | Page |
|----------------------------------|-------------------------------------|---------------------------------|-------|
| get Internet address of library. | point-to-point peer. | <i>ifDstAddrGet()</i> | 2-260 |
| library. | Point-to-Point Protocol | pppLib | 1-293 |
| routines. | Point-to-Point Protocol show | pppShow | 1-295 |
| asynchronous I/O (AIO) library | (POSIX). | aioPxLib | 1-4 |
| of asynchronous I/O operation | (POSIX). /error status | <i>aio_error()</i> | 2-9 |
| file synchronization | (POSIX). asynchronous | <i>aio_fsync()</i> | 2-10 |
| initiate asynchronous read | (POSIX). | <i>aio_read()</i> | 2-11 |
| of asynchronous I/O operation | (POSIX). /return status | <i>aio_return()</i> | 2-11 |
| asynchronous I/O request(s) | (POSIX). wait for | <i>aio_suspend()</i> | 2-12 |
| initiate asynchronous write | (POSIX). | <i>aio_write()</i> | 2-13 |
| broken-down time into string | (POSIX). convert | <i>asctime_r()</i> | 2-18 |
| clock library | (POSIX). | clockLib | 1-57 |
| get clock resolution | (POSIX). | <i>clock_getres()</i> | 2-89 |
| get current time of clock | (POSIX). | <i>clock_gettime()</i> | 2-89 |
| set clock to specified time | (POSIX). | <i>clock_settime()</i> | 2-90 |
| close directory | (POSIX). | <i>closedir()</i> | 2-91 |
| time in seconds into string | (POSIX). convert | <i>ctime_r()</i> | 2-107 |
| directory handling library | (POSIX). | dirLib | 1-80 |
| open file specified by fd | (POSIX). | <i>fdopen()</i> | 2-195 |
| return fd for stream | (POSIX). | <i>fileno()</i> | 2-201 |
| get file status information | (POSIX). | <i>fstat()</i> | 2-228 |
| get file status information | (POSIX). | <i>fstatfs()</i> | 2-229 |
| truncate file | (POSIX). | <i>ftruncate()</i> | 2-237 |
| get current default path | (POSIX). | <i>getcwd()</i> | 2-240 |
| time into broken-down time | (POSIX). convert calendar | <i>gmtime_r()</i> | 2-245 |
| send signal to task | (POSIX). | <i>kill()</i> | 2-311 |
| asynchronous I/O requests | (POSIX). initiate list of | <i>lio_listio()</i> | 2-318 |
| time into broken-down time | (POSIX). convert calendar | <i>localtime_r()</i> | 2-333 |
| specified pages into memory | (POSIX). lock | <i>mlock()</i> | 2-432 |
| used by process into memory | (POSIX). lock all pages | <i>mlockall()</i> | 2-433 |
| memory management library | (POSIX). | mmanPxLib | 1-231 |
| message queue library | (POSIX). | mqPxLib | 1-249 |
| close message queue | (POSIX). | <i>mq_close()</i> | 2-448 |
| get message queue attributes | (POSIX). | <i>mq_getattr()</i> | 2-448 |
| message is available on queue | (POSIX). notify task that | <i>mq_notify()</i> | 2-449 |
| open message queue | (POSIX). | <i>mq_open()</i> | 2-450 |
| message from message queue | (POSIX). receive | <i>mq_receive()</i> | 2-451 |
| send message to message queue | (POSIX). | <i>mq_send()</i> | 2-452 |
| set message queue attributes | (POSIX). | <i>mq_setattr()</i> | 2-453 |
| remove message queue | (POSIX). | <i>mq_unlink()</i> | 2-454 |
| unlock specified pages | (POSIX). | <i>munlock()</i> | 2-467 |
| all pages used by process | (POSIX). unlock | <i>munlockall()</i> | 2-467 |
| until time interval elapses | (POSIX). suspend current task | <i>nanosleep()</i> | 2-484 |
| open directory for searching | (POSIX). | <i>opendir()</i> | 2-543 |
| task until delivery of signal | (POSIX). suspend | <i>pause()</i> | 2-550 |
| read one entry from directory | (POSIX). | <i>readdir()</i> | 2-614 |
| position to start of directory | (POSIX). reset | <i>rewinddir()</i> | 2-631 |
| scheduling library | (POSIX). | schedPxLib | 1-319 |
| get maximum priority | (POSIX). | <i>sched_get_priority_max()</i> | 2-667 |
| get minimum priority | (POSIX). | <i>sched_get_priority_min()</i> | 2-667 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|--------------------------------|-------|
| parameters for specified task | (POSIX). get scheduling | <i>sched_getparam()</i> | 2-668 |
| get current scheduling policy | (POSIX). | <i>sched_getscheduler()</i> | 2-669 |
| get current time slice | (POSIX). | <i>sched_rr_get_interval()</i> | 2-669 |
| set task's priority | (POSIX). | <i>sched_setparam()</i> | 2-670 |
| and scheduling parameters | (POSIX). /scheduling policy | <i>sched_setscheduler()</i> | 2-671 |
| relinquish CPU | (POSIX). | <i>sched_yield()</i> | 2-671 |
| synchronization library | (POSIX). semaphore | semPxBLib | 1-344 |
| close named semaphore | (POSIX). | <i>sem_close()</i> | 2-719 |
| destroy unnamed semaphore | (POSIX). | <i>sem_destroy()</i> | 2-719 |
| get value of semaphore | (POSIX). | <i>sem_getvalue()</i> | 2-720 |
| initialize unnamed semaphore | (POSIX). | <i>sem_init()</i> | 2-721 |
| named semaphore | (POSIX). initialize/open | <i>sem_open()</i> | 2-721 |
| unlock (give) semaphore | (POSIX). | <i>sem_post()</i> | 2-723 |
| returning error if unavailable | (POSIX). /(take) semaphore, | <i>sem_trywait()</i> | 2-723 |
| remove named semaphore | (POSIX). | <i>sem_unlink()</i> | 2-724 |
| blocking if not available | (POSIX). /(take) semaphore, | <i>sem_wait()</i> | 2-725 |
| action associated with signal | (POSIX). /and/or specify | <i>sigaction()</i> | 2-742 |
| add signal to signal set | (POSIX). | <i>sigaddset()</i> | 2-743 |
| delete signal from signal set | (POSIX). | <i>sigdelset()</i> | 2-744 |
| set with no signals included | (POSIX). initialize signal | <i>sigemptyset()</i> | 2-744 |
| set with all signals included | (POSIX). initialize signal | <i>sigfillset()</i> | 2-745 |
| see if signal is in signal set | (POSIX). test to | <i>sigismember()</i> | 2-746 |
| signals blocked from delivery | (POSIX). /set of pending | <i>sigpending()</i> | 2-747 |
| and/or change signal mask | (POSIX). examine | <i>sigprocmask()</i> | 2-747 |
| task until delivery of signal | (POSIX). suspend | <i>sigsuspend()</i> | 2-749 |
| information using pathname | (POSIX). get file status | <i>stat()</i> | 2-815 |
| information using pathname | (POSIX). get file status | <i>statfs()</i> | 2-815 |
| error number to error string | (POSIX). map | <i>strerror_r()</i> | 2-821 |
| string into tokens (reentrant) | (POSIX). break down | <i>strtok_r()</i> | 2-829 |
| timer library | (POSIX). | timerLib | 1-395 |
| clock for timing base | (POSIX). /using specified | <i>timer_create()</i> | 2-912 |
| previously created timer | (POSIX). remove | <i>timer_delete()</i> | 2-913 |
| timer expiration overrun | (POSIX). return | <i>timer_getoverrun()</i> | 2-913 |
| expiration and reload value | (POSIX). /time before | <i>timer_gettime()</i> | 2-914 |
| next expiration and arm timer | (POSIX). set time until | <i>timer_settime()</i> | 2-914 |
| delete file | (POSIX). | <i>unlink()</i> | 2-946 |
| initialize | POSIX message queue library. | <i>mqPxBLibInit()</i> | 2-447 |
| | POSIX message queue show. | <i>mqPxBLibShow()</i> | 1-250 |
| facility. initialize | POSIX message queue show | <i>mqPxBLibShowInit()</i> | 2-447 |
| initialize | POSIX semaphore show facility. | <i>semPxBLibShowInit()</i> | 2-716 |
| | POSIX semaphore show library. | <i>semPxBLibShowLibInit()</i> | 1-346 |
| initialize | POSIX semaphore support. | <i>semPxBLibInit()</i> | 2-716 |
| of number raised to specified | power (ANSI). compute value | <i>pow()</i> | 2-572 |
| of number raised to specified | power (ANSI). compute value | <i>powf()</i> | 2-573 |
| (PowerPC). get | power management mode | <i>vxPowerModeGet()</i> | 2-975 |
| (PowerPC). set | power management mode | <i>vxPowerModeSet()</i> | 2-976 |
| into normalized fraction and | power of 2 (ANSI). /number | <i>frexp()</i> | 2-221 |
| multiply number by integral | power of 2 (ANSI). | <i>ldexp()</i> | 2-314 |
| library. | PPP authentication secrets | pppSecretLib | 1-295 |
| table. add secret to | PPP authentication secrets | <i>pppSecretAdd()</i> | 2-587 |

| | Keyword | Name | Page |
|-------------------------------|-------------------------------------|---------------------------------|--------|
| table. delete secret from | PPP authentication secrets | <i>pppSecretDelete()</i> | 2-588 |
| table. display | PPP authentication secrets | <i>pppSecretShow()</i> | 2-589 |
| | PPP hook library. | <i>pppHookLib</i> | 1-292 |
| get | PPP link statistics. | <i>pppstatGet()</i> | 2-589 |
| display | PPP link statistics. | <i>pppstatShow()</i> | 2-590 |
| get | PPP link status information. | <i>pppInfoGet()</i> | 2-579 |
| display | PPP link status information. | <i>pppInfoShow()</i> | 2-579 |
| delete | PPP network interface. | <i>pppDelete()</i> | 2-577 |
| initialize | PPP network interface. | <i>pppInit()</i> | 2-580 |
| routing protocol. set | priority of routes added by | <i>routeProtoPrioritySet()</i> | 2-655 |
| examine | priority of task. | <i>VXWTask::priority()</i> | 2-1032 |
| change | priority of task. | <i>VXWTask::priority()</i> | 2-1033 |
| examine | priority of task. | <i>taskPriorityGet()</i> | 2-878 |
| change | priority of task. | <i>taskPrioritySet()</i> | 2-878 |
| get maximum | priority (POSIX). | <i>sched_get_priority_max()</i> | 2-667 |
| get minimum | priority (POSIX). | <i>sched_get_priority_min()</i> | 2-667 |
| set task's | priority (POSIX). | <i>sched_setparam()</i> | 2-670 |
| initialize cache library for | processor architecture. | <i>cacheLibInit()</i> | 2-65 |
| mode (PowerPC). place | processor in reduced-power | <i>vxPowerDown()</i> | 2-975 |
| get | processor number. | <i>sysProcNumGet()</i> | 2-854 |
| set | processor number. | <i>sysProcNumSet()</i> | 2-854 |
| (SCSI-1). NCR 53C710 SCSI I/O | Processor (SIOP) library | <i>ncr710Lib</i> | 1-255 |
| (SCSI-2). NCR 53C710 SCSI I/O | Processor (SIOP) library | <i>ncr710Lib2</i> | 1-256 |
| NCR 53C8xx PCI SCSI I/O | Processor (SIOP) library/ | <i>ncr810Lib</i> | 1-257 |
| return contents of current | processor status register/ | <i>cpsr()</i> | 2-103 |
| (SPARC). return contents of | processor status register | <i>psr()</i> | 2-600 |
| determine | processor time in use (ANSI). | <i>clock()</i> | 2-88 |
| pass string to command | processor (Unimplemented)/ | <i>system()</i> | 2-859 |
| memory. flush | processor write buffers to | <i>cachePipeFlush()</i> | 2-68 |
| return contents of | program counter. | <i>pc()</i> | 2-550 |
| return contents of next | program counter (SPARC). | <i>npc()</i> | 2-534 |
| change shell | prompt. | <i>shellPromptSet()</i> | 2-741 |
| entry. display login | prompt and validate user | <i>loginPrompt()</i> | 2-341 |
| parameters. | prompt for boot line | <i>bootParamsPrompt()</i> | 2-39 |
| Protocol (ARP) client/ | proxy Address Resolution | <i>proxyLib</i> | 1-297 |
| Protocol (ARP) library. | proxy Address Resolution | <i>proxyArpLib</i> | 1-296 |
| initialize | proxy ARP. | <i>proxyArpLibInit()</i> | 2-595 |
| create | proxy ARP network. | <i>proxyNetCreate()</i> | 2-596 |
| show | proxy ARP networks. | <i>proxyNetShow()</i> | 2-597 |
| register | proxy client. | <i>proxyReg()</i> | 2-599 |
| unregister | proxy client. | <i>proxyUnreg()</i> | 2-599 |
| delete | proxy network. | <i>proxyNetDelete()</i> | 2-597 |
| | pseudo memory device driver. | <i>memDrv</i> | 1-226 |
| create | pseudo terminal. | <i>ptyDevCreate()</i> | 2-602 |
| | pseudo-terminal driver. | <i>ptyDrv</i> | 1-297 |
| initialize | pseudo-terminal driver. | <i>ptyDrv()</i> | 2-602 |
| display meaning of specified | PSR value, symbolically (ARM). | <i>psrShow()</i> | 2-601 |
| display meaning of specified | psr value, symbolically/ | <i>psrShow()</i> | 2-600 |
| compute | quotient and remainder (ANSI). | <i>div()</i> | 2-137 |
| division (ANSI). compute | quotient and remainder of | <i>ldiv()</i> | 2-315 |

| | Keyword | Name | Page |
|--|--------------------------------------|-------------------------------|-------|
| (reentrant). compute | quotient and remainder | <i>div_r()</i> | 2-138 |
| (reentrant). compute | quotient and remainder | <i>ldiv_r()</i> | 2-315 |
| initialize | R3000 cache library. | <i>cacheR3kLibInit()</i> | 2-69 |
| assembly routines. MIPS | R3000 cache management | <i>cacheR3kALib</i> | 1-47 |
| library. MIPS | R3000 cache management | <i>cacheR3kLib</i> | 1-47 |
| return size of | R3000 data cache. | <i>cacheR3kDsize()</i> | 2-69 |
| return size of | R3000 instruction cache. | <i>cacheR3kIsize()</i> | 2-69 |
| to breakpoint type (MIPS | R3000, R4000). /handler | <i>dbgBpTypeBind()</i> | 2-108 |
| initialize | R33000 cache library. | <i>cacheR33kLibInit()</i> | 2-70 |
| library. MIPS | R33000 cache management | <i>cacheR33kLib</i> | 1-48 |
| get contents of non-volatile | RAM. | <i>sysNvRamGet()</i> | 2-852 |
| write to non-volatile | RAM. | <i>sysNvRamSet()</i> | 2-853 |
| create | RAM disk device. | <i>ramDevCreate()</i> | 2-608 |
| | RAM disk driver. | <i>ramDrv</i> | 1-298 |
| (optional). prepare | RAM disk driver for use | <i>ramDrv()</i> | 2-609 |
| value of seed used to generate | random numbers (ANSI). reset | <i>srand()</i> | 2-808 |
| library. | raw block device file system | <i>rawFsLib</i> | 1-299 |
| modify mode of | raw device volume. | <i>rawFsModeChange()</i> | 2-611 |
| disable | raw device volume. | <i>rawFsVolUnmount()</i> | 2-613 |
| provide | raw I/O access. | <i>fdRawio()</i> | 2-196 |
| do | raw I/O access. | <i>ataRawio()</i> | 2-24 |
| provide | raw I/O access. | <i>ideRawio()</i> | 2-257 |
| Ethernet | raw I/O routines and hooks. | <i>etherLib</i> | 1-109 |
| associate block device with | raw volume functions. | <i>rawFsDevInit()</i> | 2-610 |
| prepare to use | raw volume library. | <i>rawFsInit()</i> | 2-611 |
| from ASCII string (ANSI). | read and convert characters | <i>sscanf()</i> | 2-808 |
| from standard input stream/ from stream (ANSI). | read and convert characters | <i>scanf()</i> | 2-666 |
| | read and convert characters | <i>fscanf()</i> | 2-222 |
| | read buffer. | <i>fioRead()</i> | 2-204 |
| device. | read bytes from file or | <i>read()</i> | 2-614 |
| tape device. | read bytes or blocks from SCSI | <i>scsiRdTape()</i> | 2-689 |
| input stream (ANSI). | read characters from standard | <i>gets()</i> | 2-242 |
| from requested NFS device. | read configuration information | <i>nfsDevInfoGet()</i> | 2-524 |
| register (MIPS). | read contents of cause | <i>intCRGet()</i> | 2-277 |
| register (MIPS). | read contents of status | <i>intSRGet()</i> | 2-283 |
| | read data into array (ANSI). | <i>fread()</i> | 2-220 |
| | read entire serial rom. | <i>dcReadAllRom()</i> | 2-112 |
| do task-level | read for tty device. | <i>tyRead()</i> | 2-932 |
| | read from PCI config space. | <i>aic7880ReadConfig()</i> | 2-6 |
| | read line with line-editing. | <i>ledRead()</i> | 2-317 |
| integer) from stream. | read next word (32-bit | <i>getw()</i> | 2-244 |
| (POSIX). | read one entry from directory | <i>readdir()</i> | 2-614 |
| initiate asynchronous | read (POSIX). | <i>aio_read()</i> | 2-11 |
| command to SCSI device and | read results. /REQUEST_SENSE | <i>scsiReqSense()</i> | 2-691 |
| block device. | read sector(s) from SCSI | <i>scsiRdSecs()</i> | 2-688 |
| characters from stream/ ROM. | read specified number of | <i>fgets()</i> | 2-201 |
| device. issue | read string from file. | <i>fioRdString()</i> | 2-203 |
| | read to PCI config space. | <i>aic7880WriteConfig()</i> | 2-7 |
| | read two bytes from serial | <i>dec21140SromWordRead()</i> | 2-114 |
| | READ_CAPACITY command to SCSI | <i>scsiReadCapacity()</i> | 2-689 |

| | Keyword | Name | Page |
|--------------------------------------|--------------------------------------|--------------------------------|--------|
| | set file read/write pointer. | <i>lseek</i> () | 2-349 |
| notify dosFs of change in | ready status. | <i>dosFsReadyChange</i> () | 2-146 |
| notify rawFsLib of change in | ready status. | <i>rawFsReadyChange</i> () | 2-612 |
| notify rt11Fs of change in | ready status. | <i>rt11FsReadyChange</i> () | 2-662 |
| notify tapeFsLib of change in | ready status. | <i>tapeFsReadyChange</i> () | 2-863 |
| check if task is | ready to run. | <i>VXWTask::isReady</i> () | 2-1030 |
| check if task is | ready to run. | <i>taskIsReady</i> () | 2-874 |
| (ANSI). | reallocate block of memory | <i>realloc</i> () | 2-615 |
| from shared mem system/ | reallocate block of memory | <i>smMemRealloc</i> () | 2-763 |
| partition. | reallocate block of memory in | <i>VXWMemPart::realloc</i> () | 2-991 |
| specified partition. | reallocate block of memory in | <i>memPartRealloc</i> () | 2-426 |
| add routine to be called at | reboot. | <i>rebootHookAdd</i> () | 2-616 |
| | reboot support library. | rebootLib | 1-303 |
| | receive data from socket. | <i>recv</i> () | 2-617 |
| socket. | receive data in zbuf from TCP | <i>zbufSockRecv</i> () | 2-1080 |
| queue. | receive message from message | <i>VXWMsgQ::receive</i> () | 2-1004 |
| queue. | receive message from message | <i>msgQReceive</i> () | 2-462 |
| queue (POSIX). | receive message from message | <i>mq_receive</i> () | 2-451 |
| | receive message from socket. | <i>recvfrom</i> () | 2-618 |
| | receive message from socket. | <i>recvmsg</i> () | 2-618 |
| UDP socket. | receive message in zbuf from | <i>zbufSockRecvfrom</i> () | 2-1081 |
| of DUART auxiliary control | register. return contents | <i>m68681Acr</i> () | 2-390 |
| in DUART auxiliary control | register. set and clear bits | <i>m68681AcrSetClr</i> () | 2-390 |
| of DUART interrupt-mask | register. /current contents | <i>m68681Imr</i> () | 2-392 |
| bits in DUART interrupt-mask | register. set and clear | <i>m68681ImrSetClr</i> () | 2-392 |
| output port configuration | register. /state of DUART | <i>m68681Opcr</i> () | 2-393 |
| output port configuration | register. /clear bits in DUART | <i>m68681OpcrSetClr</i> () | 2-394 |
| state of DUART output port | register. return current | <i>m68681Opr</i> () | 2-394 |
| bits in DUART output port | register. set and clear | <i>m68681OprSetClr</i> () | 2-395 |
| Get content of CR4 | register. | <i>pentiumCr4Get</i> () | 2-558 |
| Set specified value to CR4 | register. | <i>pentiumCr4Set</i> () | 2-558 |
| specified MSR (Model Specific | Register). get content of | <i>pentiumMsrGet</i> () | 2-559 |
| MTRR (Memory Type Range | Register). disable | <i>pentiumMtrrDisable</i> () | 2-560 |
| enable MTRR (Memory Type Range | Register). | <i>pentiumMtrrEnable</i> () | 2-560 |
| get PCMCIA configuration | register. | <i>cisConfigregGet</i> () | 2-85 |
| set PCMCIA configuration | register. | <i>cisConfigregSet</i> () | 2-85 |
| (MC680x0). return contents of | register a0 (also a1 - a7) | <i>a0</i> () | 2-1 |
| of current processor status | register (ARM). /contents | <i>cpsr</i> () | 2-103 |
| (MC680x0). return contents of | register d0 (also d1 - d7) | <i>d0</i> () | 2-108 |
| return contents of | register edi (also esi - eax) / | <i>edi</i> () | 2-150 |
| return contents of | register fp (i960). | <i>fp</i> () | 2-209 |
| (i960KB,/ return contents of | register fp0 (also fp1 - fp3) | <i>fp0</i> () | 2-209 |
| (SPARC)/ return contents of | register g0, also g1 - g7 | <i>g0</i> () | 2-238 |
| (SPARC). return contents of | register i0 (also i1 - i7) | <i>i0</i> () | 2-253 |
| return contents of status | register (i386/i486). | <i>eflags</i> () | 2-152 |
| return contents of acw | register (i960). | <i>acw</i> () | 2-4 |
| return contents of pcw | register (i960). | <i>pcw</i> () | 2-556 |
| return contents of tcw | register (i960). | <i>tcw</i> () | 2-897 |
| (SPARC). return contents of | register l0 (also l1 - l7) | <i>l0</i> () | 2-312 |
| return contents of status | register (MC680x0). | <i>sr</i> () | 2-806 |

| Keyword | Name | Page |
|--------------------------------|--|---|
| i386/i486). set task status | register (MC680x0, MIPS, <i>VXWTask::SRSet()</i> | 2-1037 |
| i386/i486). set task status | register (MC680x0, MIPS, <i>taskSRSet()</i> | 2-886 |
| read contents of cause | register (MIPS). <i>intCRGet()</i> | 2-277 |
| write contents of cause | register (MIPS). <i>intCRSet()</i> | 2-277 |
| read contents of status | register (MIPS). <i>intSRGet()</i> | 2-283 |
| update contents of status | register (MIPS). <i>intSRSet()</i> | 2-283 |
| initialize default task status | register (MIPS). <i>taskSRInit()</i> | 2-885 |
| (SPARC). return contents of | register o0 (also o1 - o7) | <i>o0()</i> 2-541 |
| return contents of | register pfp (i960). | <i>pfp()</i> 2-569 |
| | register proxy client. | <i>proxyReg()</i> 2-599 |
| (ARM). return contents of | register r0 (also r1 - r14) | <i>r0()</i> 2-606 |
| (i960). return contents of | register r3 (also r4 - r15) | <i>r3()</i> 2-607 |
| return contents of | register rip (i960). | <i>rip()</i> 2-632 |
| return contents of | register sp (i960). | <i>tsp()</i> 2-924 |
| of window invalid mask | register (SPARC). /contents | <i>wim()</i> 2-1064 |
| return contents of y | register (SPARC). | <i>y()</i> 2-1067 |
| contents of processor status | register (SPARC). return | <i>psr()</i> 2-600 |
| display dec 21040/21140 status | registers 0 thru 15. | <i>dcCsrShow()</i> 2-111 |
| set task's | registers. | <i>VXWTask::registers()</i> 2-1033 |
| display contents of task | registers. | <i>VXWTask::show()</i> 2-1035 |
| of all readable WD33C93 chip | registers. display values | <i>wd33c93Show()</i> 2-1054 |
| of task's floating-point | registers. print contents | <i>fppTaskRegsShow()</i> 2-214 |
| of all readable MB87030 SPC | registers. display values | <i>mb87030Show()</i> 2-407 |
| modify | registers. | <i>mRegs()</i> 2-454 |
| all readable NCR 53C710 SIOP | registers. display values of | <i>ncr710Show()</i> 2-491 |
| all readable NCR 53C710 SIOP | registers. display values of | <i>ncr710ShowScsi2()</i> 2-492 |
| all readable NCR 53C8xx SIOP | registers. display values of | <i>ncr810Show()</i> 2-496 |
| of all readable NCR5390 chip | registers. display values | <i>ncr5390Show()</i> 2-500 |
| (Machine Check Architecture) | registers. show MCA | <i>pentiumMcaShow()</i> 2-558 |
| specified MSR (Model Specific | Registers). set value to | <i>pentiumMsrSet()</i> 2-559 |
| set task's | registers. | <i>taskRegsSet()</i> 2-879 |
| display contents of task's | registers. | <i>taskRegsShow()</i> 2-880 |
| set hardware-dependent | registers for NCR 53C710. | <i>ncr710SetHwRegisterScsi2()</i> 2-490 |
| set hardware-dependent | registers for NCR 53C710 SIOP. | <i>ncr710SetHwRegister()</i> 2-489 |
| set hardware-dependent | registers for NCR 53C8xx SIOP. | <i>ncr810SetHwRegister()</i> 2-495 |
| get floating-point | registers from task TCB. | <i>fppTaskRegsGet()</i> 2-213 |
| get task | registers from TCB. | <i>VXWTask::registers()</i> 2-1034 |
| get task's | registers from TCB. | <i>taskRegsGet()</i> 2-879 |
| set floating-point | registers of task. | <i>fppTaskRegsSet()</i> 2-214 |
| create and initialize | release 4.x binary semaphore. | <i>semCreate()</i> 2-710 |
| library. | release 4.x binary semaphore | <i>semOLib</i> 1-344 |
| semaphore is available. take | release 4.x semaphore, if | <i>semClear()</i> 2-710 |
| device. issue | RELEASE command to SCSI | <i>scsiRelease()</i> 2-690 |
| device. issue | RELEASE UNIT command to SCSI | <i>scsiReleaseUnit()</i> 2-690 |
| compute quotient and | remainder (ANSI). | <i>div()</i> 2-137 |
| compute quotient and | remainder of division (ANSI). | <i>ldiv()</i> 2-315 |
| compute | remainder of x/y (ANSI). | <i>fmod()</i> 2-206 |
| compute | remainder of x/y (ANSI). | <i>fmodf()</i> 2-206 |
| compute quotient and | remainder (reentrant). | <i>div_r()</i> 2-138 |
| compute quotient and | remainder (reentrant). | <i>ldiv_r()</i> 2-315 |

| | Keyword | Name | Page |
|-------------------------------|-------------------------------------|--------------------------------|--------|
| | remote command library. | remLib | 1-304 |
| create | remote file device. | netDevCreate() | 2-507 |
| install network | remote file driver. | netDrv() | 2-507 |
| network | remote file I/O driver. | netDrv | 1-263 |
| log in to | remote FTP server. | ftpLogin() | 2-234 |
| exported file systems of | remote host. display | nfsExportShow() | 2-528 |
| log in to | remote host. | rlogin() | 2-644 |
| test that | remote host is reachable. | ping() | 2-570 |
| display current | remote identity. | whoami() | 2-1064 |
| VxWorks | remote login daemon. | rlogind() | 2-644 |
| initialize | remote login facility. | rloginInit() | 2-645 |
| | remote login library. | rlogLib | 1-308 |
| execute shell command on | remote machine. | rcmd() | 2-613 |
| support library. | Remote Procedure Call (RPC) | rpcLib | 1-310 |
| retrieve current time from | remote source. | sntpcTimeGet() | 2-795 |
| get file from | remote system. | tftpGet() | 2-902 |
| put file to | remote system. | tftpPut() | 2-905 |
| send TFTP message to | remote system. | tftpSend() | 2-906 |
| set | remote user name and password. | iam() | 2-254 |
| set | remote user name and password. | remCurlSet() | 2-620 |
| from RIP interface. | remove authentication hook | ripAuthHookDelete() | 2-636 |
| parameters handler. | remove configuration | dhcpcEventHookDelete() | 2-122 |
| | remove directory. | rmdir() | 2-646 |
| | remove driver from MUX. | muxDevUnload() | 2-475 |
| | remove file. | rm() | 2-645 |
| | remove file (ANSI). | remove() | 2-621 |
| list of exported file/ | remove file system from | nfsUnexport() | 2-531 |
| | remove I/O driver. | iosDrvRemove() | 2-295 |
| | remove message queue (POSIX). | mq_unlink() | 2-454 |
| (POSIX). | remove named semaphore | sem_unlink() | 2-724 |
| memory name database (VxMP/ | remove object from shared | VXWSmName::~VXWSmName() | 2-1021 |
| memory objects name database/ | remove object from shared | smNameRemove() | 2-768 |
| timer (POSIX). | remove previously created | timer_delete() | 2-913 |
| table. | remove symbol from symbol | VXWSymTab::remove() | 2-1025 |
| table. | remove symbol from symbol | symRemove() | 2-838 |
| RIP interface. | remove table bypass hook from | ripLeakHookDelete() | 2-639 |
| task. | remove task variable from | VXWTask::varDelete() | 2-1041 |
| task. | remove task variable from | taskVarDelete() | 2-892 |
| interface. | remove update filter from RIP | ripSendHookDelete() | 2-643 |
| spawn task to call function | repeatedly. | repeat() | 2-622 |
| call function | repeatedly. | repeatRun() | 2-622 |
| device and read/ issue | REQUEST_SENSE command to SCSI | scsiReqSense() | 2-691 |
| disable task | rescheduling. | taskLock() | 2-875 |
| enable task | rescheduling. | taskUnlock() | 2-890 |
| device. issue | RESERVE command to SCSI | scsiReserve() | 2-691 |
| device. issue | RESERVE UNIT command to SCSI | scsiReserveUnit() | 2-692 |
| | restart task. | VXWTask::restart() | 2-1034 |
| | restart task. | taskRestart() | 2-881 |
| coprocessor context. | restore floating-point | fppRestore() | 2-211 |
| | resume task. | VXWTask::resume() | 2-1035 |

| | Keyword | Name | Page |
|-------------------------------|-------------------------------------|----------------------------------|--------|
| | resume task. | <i>taskResume()</i> | 2-881 |
| | resume task. | <i>tr()</i> | 2-922 |
| issue | REWIND command to SCSI device. | <i>scsiRewind()</i> | 2-692 |
| number of free bytes in | ring buffer. determine | <i>VXWRingBuf::freeBytes()</i> | 2-1010 |
| get characters from | ring buffer. | <i>VXWRingBuf::get()</i> | 2-1010 |
| determine number of bytes in | ring buffer. | <i>VXWRingBuf::nBytes()</i> | 2-1012 |
| put bytes into | ring buffer. | <i>VXWRingBuf::put()</i> | 2-1012 |
| create empty | ring buffer. | <i>VXWRingBuf::VXWRingBuf()</i> | 2-1013 |
| delete | ring buffer. | <i>VXWRingBuf::~VXWRingBuf()</i> | 2-1014 |
| get characters from | ring buffer. | <i>rngBufGet()</i> | 2-646 |
| put bytes into | ring buffer. | <i>rngBufPut()</i> | 2-647 |
| create empty | ring buffer. | <i>rngCreate()</i> | 2-647 |
| delete | ring buffer. | <i>rngDelete()</i> | 2-648 |
| number of free bytes in | ring buffer. determine | <i>rngFreeBytes()</i> | 2-649 |
| determine number of bytes in | ring buffer. | <i>rngNBytes()</i> | 2-651 |
| | ring buffer class. | <i>VXWRingBuf</i> | 1-422 |
| make | ring buffer empty. | <i>VXWRingBuf::flush()</i> | 2-1009 |
| make | ring buffer empty. | <i>rngFlush()</i> | 2-648 |
| test whether | ring buffer is empty. | <i>VXWRingBuf::isEmpty()</i> | 2-1010 |
| test if | ring buffer is empty. | <i>rngIsEmpty()</i> | 2-649 |
| room). test whether | ring buffer is full (no more | <i>VXWRingBuf::isFull()</i> | 2-1011 |
| room). test if | ring buffer is full (no more | <i>rngIsFull()</i> | 2-650 |
| library. | ring buffer subroutine | <i>rngLib</i> | 1-308 |
| ring/ put byte ahead in | ring buffer without moving | <i>VXWRingBuf::putAhead()</i> | 2-1013 |
| ring/ put byte ahead in | ring buffer without moving | <i>rngPutAhead()</i> | 2-651 |
| advance | ring pointer by n bytes. | <i>VXWRingBuf::moveAhead()</i> | 2-1011 |
| advance | ring pointer by n bytes. | <i>rngMoveAhead()</i> | 2-650 |
| in ring buffer without moving | ring pointers. put byte ahead | <i>VXWRingBuf::putAhead()</i> | 2-1013 |
| in ring buffer without moving | ring pointers. put byte ahead | <i>rngPutAhead()</i> | 2-651 |
| routing table maintained by | RIP. display internal | <i>ripRouteShow()</i> | 2-641 |
| add hook to bypass | RIP and kernel routing tables. | <i>ripLeakHookAdd()</i> | 2-639 |
| interface changes. alter | RIP configuration after | <i>ripIfReset()</i> | 2-638 |
| return contents of register | rip (i960). | <i>rip()</i> | 2-632 |
| add authentication hook to | RIP interface. | <i>ripAuthHookAdd()</i> | 2-633 |
| authentication hook from | RIP interface. remove | <i>ripAuthHookDelete()</i> | 2-636 |
| remove table bypass hook from | RIP interface. | <i>ripLeakHookDelete()</i> | 2-639 |
| add update filter to | RIP interface. | <i>ripSendHookAdd()</i> | 2-642 |
| remove update filter from | RIP interface. | <i>ripSendHookDelete()</i> | 2-643 |
| terminate all | RIP processing. | <i>ripShutdown()</i> | 2-643 |
| initialize | RIP routing library. | <i>ripLibInit()</i> | 2-640 |
| Routing Information Protocol | (RIP) v1 and v2 library. | <i>ripLib</i> | 1-306 |
| read entire serial | rom. | <i>dcReadAllRom()</i> | 2-112 |
| read two bytes from serial | ROM. | <i>dec21140SromWordRead()</i> | 2-114 |
| display lines of serial | ROM for dec21140. | <i>dcViewRom()</i> | 2-112 |
| generic | ROM initialization. | <i>romStart()</i> | 2-652 |
| | ROM initialization module. | <i>bootInit</i> | 1-29 |
| (SPARC). | ROM MMU initialization | <i>mmuSparcILib</i> | 1-237 |
| transfer control to | ROM monitor. | <i>sysToMonitor()</i> | 2-860 |
| initialize MMU for | ROM (SPARC). | <i>mmuSparcRomInit()</i> | 2-434 |
| boot | ROM subroutine library. | <i>bootLib</i> | 1-30 |

| | Keyword | Name | Page |
|--------------------------------|------------------------------------|---------------------------------|--------|
| configuration module for boot | ROMs. system | bootConfig | 1-29 |
| and transfer control to boot | ROMs. reset network devices | <i>reboot()</i> | 2-616 |
| | root task. | <i>usrRoot()</i> | 2-950 |
| integer. | round number to nearest | <i>iround()</i> | 2-303 |
| integer. | round number to nearest | <i>iroundf()</i> | 2-303 |
| integer. | round number to nearest | <i>round()</i> | 2-652 |
| integer. | round number to nearest | <i>roundf()</i> | 2-653 |
| enable | round-robin selection. | <i>kernelTimeSlice()</i> | 2-310 |
| add | route. | <i>routeAdd()</i> | 2-653 |
| delete | route. | <i>routeDelete()</i> | 2-654 |
| delete | route from routing table. | <i>mRouteDelete()</i> | 2-456 |
| delete | route from routing table. | <i>mRouteEntryDelete()</i> | 2-458 |
| (OSPF Opt.). delete external | route imported into OSPF | <i>ospfExtRouteDelete()</i> | 2-546 |
| Opt.). import external | route into OSPF domain (OSPF | <i>ospfExtRouteAdd()</i> | 2-545 |
| network | route manipulation library. | routeLib | 1-309 |
| network. add | route to destination that is | <i>routeNetAdd()</i> | 2-655 |
| add protocol-specific | route to routing table. | <i>mRouteEntryAdd()</i> | 2-457 |
| OSPF version 2 (RFC 1583) | routing facilities (OSPF/ | ospfLib | 1-275 |
| (RIP) v1 and v2 library. | Routing Information Protocol | ripLib | 1-306 |
| initialize RIP | routing library. | <i>ripLibInit()</i> | 2-640 |
| function to initialize OSPF | routing (OSPF Opt.). | <i>ospfInit()</i> | 2-546 |
| priority of routes added by | routing protocol. set | <i>routeProtoPrioritySet()</i> | 2-655 |
| display | routing statistics. | <i>routeStatShow()</i> | 2-657 |
| delete route from | routing table. | <i>mRouteDelete()</i> | 2-456 |
| add protocol-specific route to | routing table. | <i>mRouteEntryAdd()</i> | 2-457 |
| delete route from | routing table. | <i>mRouteEntryDelete()</i> | 2-458 |
| print entries of | routing table. | <i>mRouteShow()</i> | 2-458 |
| get MIB-2 | routing table entry. | <i>m2IpRouteTblEntryGet()</i> | 2-367 |
| set MIB-II | routing table entry. | <i>m2IpRouteTblEntrySet()</i> | 2-368 |
| RIP. display internal | routing table maintained by | <i>ripRouteShow()</i> | 2-641 |
| hook to bypass RIP and kernel | routing tables. add | <i>ripLeakHookAdd()</i> | 2-639 |
| display host and network | routing tables. | <i>routeShow()</i> | 2-656 |
| initialize | RPC package. | <i>rpcInit()</i> | 2-657 |
| initialize task's access to | RPC package. | <i>rpcTaskInit()</i> | 2-658 |
| list contents of | RT-11 directory. | <i>lsOld()</i> | 2-349 |
| system library. | RT-11 media-compatible file | rt11FsLib | 1-311 |
| fragmented free space on | RT-11 volume. reclaim | <i>squeeze()</i> | 2-805 |
| initialize | rt11Fs device descriptor. | <i>rt11FsDevInit()</i> | 2-659 |
| initialize device and create | rt11Fs file system. | <i>rt11FsMkfs()</i> | 2-661 |
| set | rt11Fs file system date. | <i>rt11FsDateSet()</i> | 2-658 |
| prepare to use | rt11Fs library. | <i>rt11FsInit()</i> | 2-660 |
| status. notify | rt11Fs of change in ready | <i>rt11FsReadyChange()</i> | 2-662 |
| modify mode of | rt11Fs volume. | <i>rt11FsModeChange()</i> | 2-661 |
| make calling task | safe from deletion. | <i>taskSafe()</i> | 2-882 |
| partially initialize WD33C93 | SBIC structure. create and | <i>wd33c93CtrlCreate()</i> | 2-1050 |
| and partially initialize | SBIC structure. create | <i>wd33c93CtrlCreateScsi2()</i> | 2-1051 |
| user-specified fields in | SBIC structure. initialize | <i>wd33c93CtrlInit()</i> | 2-1053 |
| initialize | SCC. | <i>m68332DevInit()</i> | 2-386 |
| initialize | SCC. | <i>m68360DevInit()</i> | 2-387 |
| handle | SCC interrupt. | <i>m68332Int()</i> | 2-387 |

| | Keyword | Name | Page |
|---------------------------------------|--|---------------------------------|--------------------------|
| | handle | SCC interrupt. | <i>m68360Int()</i> 2-387 |
| Controller driver. Z8530 | SCC Serial Communications | <i>z8530Sio</i> | 1-443 |
| Motorola MC68360 | SCC UART serial driver. | <i>m68360Sio</i> | 1-211 |
| define logical partition on | SCSI block device. | <i>scsiBlkDevCreate()</i> | 2-673 |
| read sector(s) from | SCSI block device. | <i>scsiRdSecs()</i> | 2-688 |
| write sector(s) to | SCSI block device. | <i>scsiWrtSecs()</i> | 2-702 |
| pulse reset signal on | SCSI bus. | <i>scsiBusReset()</i> | 2-674 |
| WD33C93/ assert RST line on | SCSI bus (Western Digital | <i>sysScsiBusReset()</i> | 2-855 |
| system | SCSI configuration. | <i>sysScsiConfig()</i> | 2-855 |
| all devices connected to | SCSI controller. configure | <i>scsiAutoConfig()</i> | 2-672 |
| physical devices attached to | SCSI controller. list | <i>scsiShow()</i> | 2-695 |
| (SCSI-1). NCR 53C90 Advanced | SCSI Controller (ASC) library | <i>ncr5390Lib1</i> | 1-258 |
| (SCSI-2). NCR 53C90 Advanced | SCSI Controller (ASC) library | <i>ncr5390Lib2</i> | 1-259 |
| notify SCSI manager of | SCSI (controller) event. | <i>scsiMgrEventNotify()</i> | 2-682 |
| send event to | SCSI controller state machine. | <i>scsiMgrCtrlEvent()</i> | 2-681 |
| enable double speed | SCSI data transfers. | <i>aic7880EnableFast20()</i> | 2-5 |
| issue ERASE command to | SCSI device. | <i>scsiErase()</i> | 2-677 |
| issue FORMAT UNIT command to | SCSI device. | <i>scsiFormatUnit()</i> | 2-677 |
| issue INQUIRY command to | SCSI device. | <i>scsiInquiry()</i> | 2-679 |
| issue LOAD/UNLOAD command to | SCSI device. | <i>scsiLoadUnit()</i> | 2-680 |
| issue MODE_SELECT command to | SCSI device. | <i>scsiModeSelect()</i> | 2-684 |
| issue MODE_SENSE command to | SCSI device. | <i>scsiModeSense()</i> | 2-684 |
| READ_CAPACITY command to | SCSI device. issue | <i>scsiReadCapacity()</i> | 2-689 |
| issue RELEASE command to | SCSI device. | <i>scsiRelease()</i> | 2-690 |
| issue RELEASE UNIT command to | SCSI device. | <i>scsiReleaseUnit()</i> | 2-690 |
| issue RESERVE command to | SCSI device. | <i>scsiReserve()</i> | 2-691 |
| issue RESERVE UNIT command to | SCSI device. | <i>scsiReserveUnit()</i> | 2-692 |
| issue REWIND command to | SCSI device. | <i>scsiRewind()</i> | 2-692 |
| command to | SCSI device. READ_BLOCK_LIMITS .. | <i>scsiSeqReadBlockLimits()</i> | 2-694 |
| tape on specified physical | SCSI device. move | <i>scsiSpace()</i> | 2-696 |
| START_STOP UNIT command to | SCSI device. issue | <i>scsiStartStopUnit()</i> | 2-696 |
| TEST_UNIT_READY command to | SCSI device. issue | <i>scsiTestUnitRdy()</i> | 2-700 |
| issue REQUEST_SENSE command to | SCSI device and read results. | <i>scsiReqSense()</i> | 2-691 |
| File. Adaptec 7880 | SCSI Host Adapter Library | <i>aic7880Lib</i> | 1-1 |
| | SCSI initialization. | <i>usrScsi</i> | 1-413 |
| library (SCSI-1). NCR 53C710 | SCSI I/O Processor (SIOP) | <i>ncr710Lib</i> | 1-255 |
| library (SCSI-2). NCR 53C710 | SCSI I/O Processor (SIOP) | <i>ncr710Lib2</i> | 1-256 |
| library/ NCR 53C8xx PCI | SCSI I/O Processor (SIOP) | <i>ncr810Lib</i> | 1-257 |
| Computer System Interface | (SCSI) library. Small | <i>scsiLib</i> | 1-332 |
| for all devices (SCSI-2). | SCSI library common commands | <i>scsiCommonLib</i> | 1-330 |
| devices (SCSI-2). | SCSI library for direct access | <i>scsiDirectLib</i> | 1-331 |
| /Computer System Interface | (SCSI) library (SCSI-1). | <i>scsi1Lib</i> | 1-320 |
| /Computer System Interface | (SCSI) library (SCSI-2). | <i>scsi2Lib</i> | 1-324 |
| initialize fields in | SCSI logical partition. | <i>scsiBlkDevInit()</i> | 2-673 |
| show status information for | SCSI manager. | <i>scsiMgrShow()</i> | 2-682 |
| | SCSI manager library (SCSI-2). | <i>scsiMgrLib</i> | 1-333 |
| (controller) event. notify | SCSI manager of SCSI | <i>scsiMgrEventNotify()</i> | 2-682 |
| perform post-processing after | SCSI message is sent. | <i>scsiMsgOutComplete()</i> | 2-685 |
| target. handle complete | SCSI message received from | <i>scsiMsgInComplete()</i> | 2-685 |
| configure | SCSI peripherals. | <i>usrScsiConfig()</i> | 2-951 |

| | Keyword | Name | Page |
|-------------------------------------|---|--------------------------------|--------|
| structure. create | SCSI physical device | <i>scsiPhysDevCreate()</i> | 2-686 |
| structure. delete | SCSI physical-device | <i>scsiPhysDevDelete()</i> | 2-687 |
| initialize on-board | SCSI port. | <i>sysScsiInit()</i> | 2-857 |
| library. Fujitsu MB87030 | SCSI Protocol Controller (SPC) | <i>mb87030Lib</i> | 1-223 |
| library (SCSI-2). | SCSI sequential access device | <i>scsiSeqLib</i> | 1-334 |
| create | SCSI sequential device. | <i>scsiSeqDevCreate()</i> | 2-693 |
| write file marks to | SCSI sequential device. | <i>scsiWrtFileMarks()</i> | 2-701 |
| read bytes or blocks from | SCSI tape device. | <i>scsiRdTape()</i> | 2-689 |
| issue MODE_SELECT command to | SCSI tape device. | <i>scsiTapeModeSelect()</i> | 2-697 |
| issue MODE_SENSE command to | SCSI tape device. | <i>scsiTapeModeSense()</i> | 2-698 |
| write data to | SCSI tape device. | <i>scsiWrtTape()</i> | 2-702 |
| get options for one or all | SCSI targets. | <i>scsiTargetOptionsGet()</i> | 2-698 |
| set options for one or all | SCSI targets. | <i>scsiTargetOptionsSet()</i> | 2-699 |
| caches is disabled. inform | SCSI that hardware snooping of | <i>scsiCacheSnoopDisable()</i> | 2-675 |
| caches is enabled. inform | SCSI that hardware snooping of | <i>scsiCacheSnoopEnable()</i> | 2-675 |
| perform generic | SCSI thread initialization. | <i>scsiThreadInit()</i> | 2-700 |
| library (SCSI-2). | SCSI thread-level controller | <i>scsiCtrlLib</i> | 1-330 |
| I/O Processor (SIOP) library | (SCSI-1). NCR 53C710 SCSI | <i>ncr710Lib</i> | 1-255 |
| SCSI Controller (ASC) library | (SCSI-1). NCR 53C90 Advanced | <i>ncr5390Lib1</i> | 1-258 |
| Interface (SCSI) library | (SCSI-1). /Computer System | <i>scsi1Lib</i> | 1-320 |
| Interface Controller library | (SCSI-1). WD33C93 SCSI-Bus | <i>wd33c93Lib1</i> | 1-431 |
| I/O Processor (SIOP) library | (SCSI-2). NCR 53C710 SCSI | <i>ncr710Lib2</i> | 1-256 |
| I/O Processor (SIOP) library | (SCSI-2). NCR 53C8xx PCI SCSI | <i>ncr810Lib</i> | 1-257 |
| SCSI Controller (ASC) library | (SCSI-2). NCR 53C90 Advanced | <i>ncr5390Lib2</i> | 1-259 |
| Interface (SCSI) library | (SCSI-2). /Computer System | <i>scsi2Lib</i> | 1-324 |
| commands for all devices | (SCSI-2). SCSI library common | <i>scsiCommonLib</i> | 1-330 |
| controller library | (SCSI-2). SCSI thread-level | <i>scsiCtrlLib</i> | 1-330 |
| for direct access devices | (SCSI-2). SCSI library | <i>scsiDirectLib</i> | 1-331 |
| SCSI manager library | (SCSI-2). | <i>scsiMgrLib</i> | 1-333 |
| access device library | (SCSI-2). SCSI sequential | <i>scsiSeqLib</i> | 1-334 |
| Interface Controller library | (SCSI-2). WD33C93 SCSI-Bus | <i>wd33c93Lib2</i> | 1-431 |
| initialize | SCSI-2 interface to <i>scsiLib</i> | <i>scsi2IfInit()</i> | 2-672 |
| return pointer to | SCSI_PHYS_DEV structure. | <i>scsiPhysDevIdGet()</i> | 2-687 |
| wake up task pended in | <i>select()</i> | <i>selWakeup()</i> | 2-705 |
| get type of | <i>select()</i> wake-up node. | <i>selWakeupType()</i> | 2-707 |
| add wake-up node to | <i>select()</i> wake-up list. | <i>selNodeAdd()</i> | 2-704 |
| find and delete node from | <i>select()</i> wake-up list. | <i>selNodeDelete()</i> | 2-705 |
| wake up all tasks in | <i>select()</i> wake-up list. | <i>selWakeupAll()</i> | 2-706 |
| initialize | <i>select()</i> wake-up list. | <i>selWakeupListInit()</i> | 2-706 |
| get number of nodes in | <i>select()</i> wake-up list. | <i>selWakeupListLen()</i> | 2-707 |
| initialize | select facility. | <i>selectInit()</i> | 2-704 |
| UNIX BSD 4.3 | select library. | <i>selectLib</i> | 1-335 |
| unblock every task pended on | semaphore. | <i>VXWSem::flush()</i> | 2-1014 |
| give | semaphore. | <i>VXWSem::give()</i> | 2-1015 |
| task IDs that are blocked on | semaphore. get list of | <i>VXWSem::info()</i> | 2-1016 |
| show information about | semaphore. | <i>VXWSem::show()</i> | 2-1016 |
| take | semaphore. | <i>VXWSem::take()</i> | 2-1017 |
| delete | semaphore. | <i>VXWSem::~VXWSem()</i> | 2-1018 |
| create and initialize binary | semaphore. | <i>semBCreate()</i> | 2-708 |
| create and initialize counting | semaphore. | <i>semCCreate()</i> | 2-709 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|------------------------------|--------|
| initialize release 4.x binary | semaphore. create and | <i>semCreate()</i> | 2-710 |
| delete | semaphore. | <i>semDelete()</i> | 2-711 |
| unblock every task pended on | semaphore. | <i>semFlush()</i> | 2-712 |
| give | semaphore. | <i>semGive()</i> | 2-712 |
| task IDs that are blocked on | semaphore. get list of | <i>semInfo()</i> | 2-713 |
| initialize static binary | semaphore. | <i>semInit()</i> | 2-714 |
| initialize mutual-exclusion | semaphore. create and | <i>semMCreate()</i> | 2-714 |
| show information about | semaphore. | <i>semShow()</i> | 2-717 |
| take | semaphore. | <i>semTake()</i> | 2-718 |
| create and initialize binary | semaphore. | <i>VXWBSem::VXWBSem()</i> | 2-978 |
| create and initialize counting | semaphore. | <i>VXWCSem::VXWCSem()</i> | 2-980 |
| initialize mutual-exclusion | semaphore. create and | <i>VXWMSem::VXWMSem()</i> | 2-999 |
| available/ lock (take) | semaphore, blocking if not | <i>sem_wait()</i> | 2-725 |
| | semaphore classes. | <i>VXWSem</i> | 1-423 |
| reveal underlying | semaphore ID. | <i>VXWSem::id()</i> | 2-1015 |
| build semaphore object from | semaphore ID. | <i>VXWSem::VXWSem()</i> | 2-1018 |
| available. take release 4.x | semaphore, if semaphore is | <i>semClear()</i> | 2-710 |
| take release 4.x semaphore, if | semaphore is available. | <i>semClear()</i> | 2-710 |
| binary | semaphore library. | <i>semBLib</i> | 1-336 |
| counting | semaphore library. | <i>semCLib</i> | 1-338 |
| general | semaphore library. | <i>semLib</i> | 1-339 |
| mutual-exclusion | semaphore library. | <i>semMLib</i> | 1-341 |
| release 4.x binary | semaphore library. | <i>semOLib</i> | 1-344 |
| shared memory | semaphore library (VxMP Opt.). | <i>semSmLib</i> | 1-347 |
| semaphore ID. build | semaphore object from | <i>VXWSem::VXWSem()</i> | 2-1018 |
| close named | semaphore (POSIX). | <i>sem_close()</i> | 2-719 |
| destroy unnamed | semaphore (POSIX). | <i>sem_destroy()</i> | 2-719 |
| get value of | semaphore (POSIX). | <i>sem_getvalue()</i> | 2-720 |
| initialize unnamed | semaphore (POSIX). | <i>sem_init()</i> | 2-721 |
| initialize/open named | semaphore (POSIX). | <i>sem_open()</i> | 2-721 |
| unlock (give) | semaphore (POSIX). | <i>sem_post()</i> | 2-723 |
| remove named | semaphore (POSIX). | <i>sem_unlink()</i> | 2-724 |
| unavailable/ lock (take) | semaphore, returning error if | <i>sem_trywait()</i> | 2-723 |
| initialize POSIX | semaphore show facility. | <i>semPxShowInit()</i> | 2-716 |
| initialize | semaphore show facility. | <i>semShowInit()</i> | 2-717 |
| POSIX | semaphore show library. | <i>semPxShow</i> | 1-346 |
| | semaphore show routines. | <i>semShow</i> | 1-346 |
| initialize POSIX | semaphore support. | <i>semPxLibInit()</i> | 2-716 |
| library (POSIX). | semaphore synchronization | <i>semPxLib</i> | 1-344 |
| /shared memory binary | semaphore (VxMP Opt.). | <i>semBSmCreate()</i> | 2-708 |
| /shared memory counting | semaphore (VxMP Opt.). | <i>semCSmCreate()</i> | 2-710 |
| give mutual-exclusion | semaphore without/ | <i>semMGiveForce()</i> | 2-715 |
| give mutual-exclusion | semaphore without/ | <i>VXWMSem::giveForce()</i> | 2-999 |
| File Transfer Protocol (FTP) | server. | <i>ftpdLib</i> | 1-120 |
| retrieve current DHCP | server. | <i>dhcpcServerGet()</i> | 2-129 |
| display current DHCP | server. | <i>dhcpcServerShow()</i> | 2-129 |
| address storage hook for | server. assign permanent | <i>dhcpsAddressHookAdd()</i> | 2-132 |
| lease storage hook for | server. assign permanent | <i>dhcpsLeaseHookAdd()</i> | 2-135 |
| log in to remote FTP | server. | <i>ftpLogin()</i> | 2-234 |
| initialize NFS | server. | <i>nfsdInit()</i> | 2-525 |

| | Keyword | Name | Page |
|--------------------------------|--|------------------------------|--------|
| get status of NFS | server. | <i>nfsdStatusGet()</i> | 2-527 |
| show status of NFS | server. | <i>nfsdStatusShow()</i> | 2-527 |
| set TFTP | server address. | <i>tftpPeerSet()</i> | 2-904 |
| change SNTP | server broadcast settings. | <i>sntpsConfigSet()</i> | 2-797 |
| TFTP | server daemon task. | <i>tftpdTask()</i> | 2-901 |
| address. query DNS | server for host name of IP | <i>resolvGetHostByAddr()</i> | 2-624 |
| query DNS | server for IP address of host. | <i>resolvGetHostByName()</i> | 2-625 |
| Network File System (NFS) | server library. | <i>nfsdLib</i> | 1-266 |
| Network Time Protocol (SNTP) | server library. Simple | <i>sntpsLib</i> | 1-367 |
| telnet | server library. | <i>telnetLib</i> | 1-392 |
| Trivial File Transfer Protocol | server library. | <i>tftpdLib</i> | 1-392 |
| Configuration Protocol (DHCP) | server library. Dynamic Host | <i>dhcpsLib</i> | 1-75 |
| get control connection to FTP | server on specified host. | <i>ftpHookup()</i> | 2-233 |
| structures. set up DHCP | server parameters and data | <i>dhcpsInit()</i> | 2-133 |
| terminate FTP | server task. | <i>ftpdDelete()</i> | 2-232 |
| initialize FTP | server task. | <i>ftpdInit()</i> | 2-232 |
| initialize TFTP | server task. | <i>tftpdInit()</i> | 2-901 |
| ANSI | setjmp documentation. | <i>ansiSetjmp</i> | 1-15 |
| /pages to virtual space in | shared global virtual mem/ | <i>vmGlobalMap()</i> | 2-963 |
| (VxMP/ /block of memory from | shared mem system partition | <i>smMemRealloc()</i> | 2-763 |
| network interface driver. | shared memory backplane | <i>if_sm</i> | 1-166 |
| (VxMP/ create and initialize | shared memory binary semaphore | <i>semBsmCreate()</i> | 2-708 |
| naming behavior common to all | shared memory classes. | <i>VXWSmName</i> | 1-425 |
| create and initialize | shared memory counting/ | <i>semCsmCreate()</i> | 2-710 |
| library (VxMP Opt.). | shared memory management | <i>smMemLib</i> | 1-355 |
| routines (VxMP Opt.). | shared memory management show | <i>smMemShow</i> | 1-357 |
| library (VxMP Opt.). | shared memory message queue | <i>msgQSmLib</i> | 1-252 |
| (VxMP/ create and initialize | shared memory message queue | <i>msgQSmCreate()</i> | 2-466 |
| (VxMP/ remove object from | shared memory name database <i>VXWSmName::~VXWSmName()</i> | | 2-1021 |
| (VxMP Opt.). add name to | shared memory name database | <i>smNameAdd()</i> | 2-765 |
| show information about | shared memory network. | <i>smNetShow()</i> | 2-772 |
| VxWorks interface to | shared memory network/ | <i>smNetLib</i> | 1-360 |
| initialize | shared memory network driver. | <i>smNetInit()</i> | 2-771 |
| show routines. | shared memory network driver | <i>smNetShow</i> | 1-361 |
| interface. attach | shared memory network | <i>smNetAttach()</i> | 2-769 |
| get address associated with | shared memory network/ | <i>smNetInetGet()</i> | 2-770 |
| (VxMP Opt.). look up | shared memory object by name | <i>smNameFind()</i> | 2-766 |
| (VxMP Opt.). look up | shared memory object by value | <i>smNameFindByValue()</i> | 2-767 |
| initialization. | shared memory object | <i>usrSmObj</i> | 1-413 |
| Opt.). get name and type of | shared memory object (VxMP | <i>VXWSmName::nameGet()</i> | 2-1019 |
| Opt.). get name of | shared memory object (VxMP | <i>VXWSmName::nameGet()</i> | 2-1019 |
| initialize | shared memory objects. | <i>usrSmObjInit()</i> | 2-951 |
| descriptor (VxMP/ initialize | shared memory objects | <i>smObjInit()</i> | 2-774 |
| (VxMP/ attach calling CPU to | shared memory objects facility | <i>smObjAttach()</i> | 2-772 |
| (VxMP Opt.). install | shared memory objects facility | <i>smObjLibInit()</i> | 2-775 |
| (VxMP Opt.). initialize | shared memory objects facility | <i>smObjSetup()</i> | 2-776 |
| (VxMP Opt.). | shared memory objects library | <i>smObjLib</i> | 1-361 |
| database library (VxMP Opt.). | shared memory objects name | <i>smNameLib</i> | 1-357 |
| database show routines (VxMP/ | shared memory objects name | <i>smNameShow</i> | 1-360 |
| database/ remove object from | shared memory objects name | <i>smNameRemove()</i> | 2-768 |

| | Keyword | Name | Page |
|--|---------------------------------------|-----------------------------|--------|
| database/ show contents of routines (VxMP Opt.). | shared memory objects name | <i>smNameShow()</i> | 2-768 |
| display current status of Opt.). create library (VxMP Opt.). | shared memory objects show | <i>smObjShow</i> | 1-364 |
| block of memory (VxMP/ free blocks and statistics/ show (VxMP Opt.). add memory to allocate memory for array from find largest free block in allocate block of memory from (VxMP/ set debug options for (VxMP/ define name string in start | shared memory objects (VxMP/ | <i>smObjShow()</i> | 2-777 |
| lock access to machine. execute | shared memory partition (VxMP | <i>memPartSmCreate()</i> | 2-427 |
| display or set size of display or set size of change script. signal | shared memory semaphore | <i>semSmLib</i> | 1-347 |
| input/output/error fds. set POSIX message queue | shared memory system partition | <i>smMemFree()</i> | 2-761 |
| PCIC chip. | shared memory system partition | <i>smMemShow()</i> | 2-764 |
| PCMCIA chip. | shared memory system partition | <i>smMemAddToPool()</i> | 2-759 |
| TCIC chip. | shared memory system partition/ | <i>smMemCalloc()</i> | 2-760 |
| specified physical device. | shared memory system partition/ | <i>smMemFindMax()</i> | 2-760 |
| objects name database (VxMP/ loaded modules. | shared memory system partition/ | <i>smMemMalloc()</i> | 2-762 |
| initialize watchdog | shared memory system partition | <i>smMemOptionsSet()</i> | 2-762 |
| initialize DHCP | shared-memory name database | <i>VXWSmName::nameSet()</i> | 2-1020 |
| initialize floating-point | shell. | <i>shellInit()</i> | 2-739 |
| initialize I/O system | shell. | <i>shellLock()</i> | 2-740 |
| initialize memory partition | shell command on remote | <i>rcmd()</i> | 2-613 |
| initialize POSIX message queue | shell entry point. | <i>shell()</i> | 2-738 |
| initialize message queue | shell execution routines. | <i>shellLib</i> | 1-348 |
| initialize POSIX semaphore | shell history. | <i>h()</i> | 2-246 |
| initialize semaphore | shell history. | <i>shellHistory()</i> | 2-739 |
| initialize standard I/O | shell prompt. | <i>shellPromptSet()</i> | 2-741 |
| initialize task hook | shell to stop processing | <i>shellScriptAbort()</i> | 2-741 |
| include virtual memory | shell's default | <i>shellOrigStdSet()</i> | 2-740 |
| queue. | show. | <i>mqPxShow</i> | 1-250 |
| queue. | show AIO requests. | <i>aioShow()</i> | 2-8 |
| semaphore. | show all configurations of | <i>pcicShow()</i> | 2-554 |
| semaphore. | show all configurations of | <i>pcmciaShow()</i> | 2-555 |
| | show all configurations of | <i>tcicShow()</i> | 2-896 |
| | show ATA/IDE disk parameters. | <i>ataShow()</i> | 2-24 |
| | show BLK_DEV structures on | <i>scsiBlkDevShow()</i> | 2-674 |
| | show CIS information. | <i>cisShow()</i> | 2-87 |
| | show contents of shared memory | <i>smNameShow()</i> | 2-768 |
| | show current status for all | <i>moduleShow()</i> | 2-443 |
| | show enabled ports. | <i>proxyPortShow()</i> | 2-598 |
| | show facility. | <i>wdShowInit()</i> | 2-1062 |
| | show facility. | <i>dhcpcShowInit()</i> | 2-130 |
| | show facility. | <i>fppShowInit()</i> | 2-213 |
| | show facility. | <i>iosShowInit()</i> | 2-297 |
| | show facility. | <i>memShowInit()</i> | 2-430 |
| | show facility. | <i>mqPxShowInit()</i> | 2-447 |
| | show facility. | <i>msgQShowInit()</i> | 2-465 |
| | show facility. | <i>semPxShowInit()</i> | 2-716 |
| | show facility. | <i>semShowInit()</i> | 2-717 |
| | show facility. | <i>stdioShowInit()</i> | 2-817 |
| | show facility. | <i>taskHookShowInit()</i> | 2-870 |
| | show facility (VxVMI Opt.). | <i>vmShowInit()</i> | 2-968 |
| | show information about message | <i>VXWMsgQ::show()</i> | 2-1007 |
| | show information about message | <i>msgQShow()</i> | 2-464 |
| | show information about | <i>VXWSem::show()</i> | 2-1016 |
| | show information about | <i>semShow()</i> | 2-717 |

| | Keyword | Name | Page |
|--------------------------------|--|-------------------------------------|--------------------------|
| | memory network. | show information about shared | <i>smNetShow()</i> 2-772 |
| | watchdog. | show information about | <i>wdShow()</i> 2-1062 |
| PCMCIA host bus adaptor chip | show library. Intel 82365SL | <i>pcicShow</i> | 1-281 |
| PCMCIA | show library. | <i>pcmciaShow</i> | 1-282 |
| POSIX semaphore | show library. | <i>semPxShow</i> | 1-346 |
| PCMCIA host bus adaptor chip | show library. Databook TCIC/2 | <i>tcicShow</i> | 1-391 |
| PCMCIA CIS | show library. | <i>cisShow</i> | 1-57 |
| asynchronous I/O (AIO) | show library. | <i>aioPxShow</i> | 1-8 |
| | routines. show list of task create | <i>taskCreateHookShow()</i> | 2-866 |
| | routines. show list of task delete | <i>taskDeleteHookShow()</i> | 2-869 |
| | routines. show list of task switch | <i>taskSwitchHookShow()</i> | 2-889 |
| | show LPT statistics. | <i>lptShow()</i> | 2-348 |
| Architecture) registers. | show MCA (Machine Check | <i>pentiumMcaShow()</i> | 2-558 |
| | statistics. show network stack data pool | <i>netStackDataPoolShow()</i> | 2-519 |
| | statistics. show network stack system pool | <i>netStackSysPoolShow()</i> | 2-520 |
| | statistics. show partition blocks and | <i>memPartShow()</i> | 2-427 |
| | statistics. show partition blocks and | <i>VXWMemPart::show()</i> | 2-992 |
| Monitoring Counters). | show PMCs (Performance | <i>pentiumPmcShow()</i> | 2-564 |
| | show pool statistics. | <i>netPoolShow()</i> | 2-518 |
| | show proxy ARP networks. | <i>proxyNetShow()</i> | 2-597 |
| and PCMCIA) disk device driver | show routine. ATA/IDE (LOCAL | <i>ataShow</i> | 1-27 |
| initialize ATA/IDE disk driver | show routine. | <i>ataShowInit()</i> | 2-25 |
| initialize task | show routine facility. | <i>taskShowInit()</i> | 2-883 |
| floating-point | show routines. | <i>fppShow</i> | 1-120 |
| I/O system | show routines. | <i>iosShow</i> | 1-181 |
| memory | show routines. | <i>memShow</i> | 1-231 |
| message queue | show routines. | <i>msgQShow</i> | 1-252 |
| and PentiumPro specific | show routines. Pentium | <i>pentiumShow</i> | 1-288 |
| Point-to-Point Protocol | show routines. | <i>pppShow</i> | 1-295 |
| semaphore | show routines. | <i>semShow</i> | 1-346 |
| shared memory network driver | show routines. | <i>smNetShow</i> | 1-361 |
| task hook | show routines. | <i>taskHookShow</i> | 1-386 |
| task | show routines. | <i>taskShow</i> | 1-389 |
| watchdog | show routines. | <i>wdShow</i> | 1-441 |
| initialize ICMP | show routines. | <i>icmpShowInit()</i> | 2-255 |
| initialize IGMP | show routines. | <i>igmpShowInit()</i> | 2-267 |
| initialize network | show routines. | <i>netShowInit()</i> | 2-519 |
| initialize TCP | show routines. | <i>tcpShowInit()</i> | 2-897 |
| initialize UDP | show routines. | <i>udpShowInit()</i> | 2-933 |
| drivers. initialize all | show routines for PCMCIA | <i>pcmciaShowInit()</i> | 2-556 |
| shared memory management | show routines (VxMP Opt.). | <i>smMemShow</i> | 1-357 |
| /memory objects name database | show routines (VxMP Opt.). | <i>smNameShow</i> | 1-360 |
| shared memory objects | show routines (VxMP Opt.). | <i>smObjShow</i> | 1-364 |
| virtual memory | show routines (VxVMI Opt.). | <i>vmShow</i> | 1-416 |
| partition blocks and/ | show shared memory system | <i>smMemShow()</i> | 2-764 |
| physical device. | show status information for | <i>scsiPhysDevShow()</i> | 2-688 |
| SCSI manager. | show status information for | <i>scsiMgrShow()</i> | 2-682 |
| | show status of NFS server. | <i>nfsdStatusShow()</i> | 2-527 |
| blocks and statistics. | show system memory partition | <i>memShow()</i> | 2-429 |
| information. | show volume configuration | <i>cdromFsVolConfigShow()</i> | 2-82 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------|--------------------------------|--------|
| | shut down network connection. | <i>shutdown()</i> | 2-742 |
| handler associated with | signal. specify | <i>signal()</i> | 2-746 |
| wait for | signal. | <i>sigtimedwait()</i> | 2-750 |
| connect user routine to timer | signal. | <i>timer_connect()</i> | 2-911 |
| initialize | signal facilities. | <i>sigInit()</i> | 2-745 |
| initialize queued | signal facilities. | <i>sigqueueInit()</i> | 2-748 |
| software | signal facility library. | <i>sigLib</i> | 1-349 |
| (POSIX). delete | signal from signal set | <i>sigdelset()</i> | 2-744 |
| install | signal handler. | <i>sigvec()</i> | 2-751 |
| (POSIX). test to see if | signal is in signal set | <i>sigismember()</i> | 2-746 |
| set | signal mask. | <i>sigsetmask()</i> | 2-749 |
| examine and/or change | signal mask (POSIX). | <i>sigprocmask()</i> | 2-747 |
| pulse reset | signal on SCSI bus. | <i>scsiBusReset()</i> | 2-674 |
| suspend task until delivery of | signal (POSIX). | <i>pause()</i> | 2-550 |
| specify action associated with | signal (POSIX). /and/or | <i>sigaction()</i> | 2-742 |
| suspend task until delivery of | signal (POSIX). | <i>sigsuspend()</i> | 2-749 |
| add signal to | signal set (POSIX). | <i>sigaddset()</i> | 2-743 |
| delete signal from | signal set (POSIX). | <i>sigdelset()</i> | 2-744 |
| test to see if signal is in | signal set (POSIX). | <i>sigismember()</i> | 2-746 |
| included (POSIX). initialize | signal set with all signals | <i>sigfillset()</i> | 2-745 |
| included (POSIX). initialize | signal set with no signals | <i>sigemptyset()</i> | 2-744 |
| processing script. | signal shell to stop | <i>shellScriptAbort()</i> | 2-741 |
| send | signal to caller's task. | <i>raise()</i> | 2-607 |
| add | signal to signal set (POSIX). | <i>sigaddset()</i> | 2-743 |
| send | signal to task. | <i>VXWTask::kill()</i> | 2-1030 |
| send queued | signal to task. | <i>VXWTask::sigqueue()</i> | 2-1037 |
| send queued | signal to task. | <i>sigqueue()</i> | 2-748 |
| send | signal to task (POSIX). | <i>kill()</i> | 2-311 |
| add to set of blocked | signals. | <i>sigblock()</i> | 2-743 |
| wait for real-time | signals. | <i>sigwaitinfo()</i> | 2-752 |
| retrieve set of pending | signals blocked from delivery/ | <i>sigpending()</i> | 2-747 |
| initialize signal set with no | signals included (POSIX). | <i>sigemptyset()</i> | 2-744 |
| initialize signal set with all | signals included (POSIX). | <i>sigfillset()</i> | 2-745 |
| compute both | sine and cosine. | <i>sincos()</i> | 2-753 |
| compute both | sine and cosine. | <i>sincosf()</i> | 2-754 |
| compute arc | sine (ANSI). | <i>asin()</i> | 2-19 |
| compute arc | sine (ANSI). | <i>asinf()</i> | 2-19 |
| compute | sine (ANSI). | <i>sin()</i> | 2-753 |
| compute | sine (ANSI). | <i>sinf()</i> | 2-754 |
| compute hyperbolic | sine (ANSI). | <i>sinh()</i> | 2-755 |
| compute hyperbolic | sine (ANSI). | <i>sinhf()</i> | 2-755 |
| perform | single-step. | <i>ncr710SingleStep()</i> | 2-501 |
| enable/disable script | single-step. | <i>ncr710StepEnable()</i> | 2-501 |
| subroutine. | single-step, but step over | <i>so()</i> | 2-798 |
| | single-step task. | <i>s()</i> | 2-662 |
| with serial channel. get | SIO_CHAN device associated | <i>sysSerialChanGet()</i> | 2-857 |
| structure for NCR 53C710 | SIOP. create control | <i>ncr710CtrlCreate()</i> | 2-485 |
| structure for NCR 53C710 | SIOP. create control | <i>ncr710CtrlCreateScsi2()</i> | 2-486 |
| structure for NCR 53C710 | SIOP. initialize control | <i>ncr710CtrlInit()</i> | 2-487 |
| structure for NCR 53C710 | SIOP. initialize control | <i>ncr710CtrlInitScsi2()</i> | 2-488 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|--------------------------------|--------|
| registers for NCR 53C710 | SIOP. set hardware-dependent | <i>ncr710SetHwRegister()</i> | 2-489 |
| structure for NCR 53C8xx | SIOP. create control | <i>ncr810CtrlCreate()</i> | 2-493 |
| structure for NCR 53C8xx | SIOP. initialize control | <i>ncr810CtrlInit()</i> | 2-494 |
| registers for NCR 53C8xx | SIOP. set hardware-dependent | <i>ncr810SetHwRegister()</i> | 2-495 |
| NCR 53C710 SCSI I/O Processor | (SIOP) library (SCSI-1). | <i>ncr710Lib</i> | 1-255 |
| NCR 53C710 SCSI I/O Processor | (SIOP) library (SCSI-2). | <i>ncr710Lib2</i> | 1-256 |
| 53C8xx PCI SCSI I/O Processor | (SIOP) library (SCSI-2). NCR | <i>ncr810Lib</i> | 1-257 |
| of all readable NCR 53C710 | SIOP registers. /values | <i>ncr710Show()</i> | 2-491 |
| of all readable NCR 53C710 | SIOP registers. /values | <i>ncr710ShowScsi2()</i> | 2-492 |
| of all readable NCR 53C8xx | SIOP registers. /values | <i>ncr810Show()</i> | 2-496 |
| for specified buffer | size. return <i>CL_POOL_ID</i> | <i>netCIPoolIdGet()</i> | 2-506 |
| return page | size. | <i>vmBasePageSizeGet()</i> | 2-958 |
| block. find | size of largest available free | <i>memPartFindMax()</i> | 2-424 |
| block. find | size of largest available free | <i>VXWMemPart:findMax()</i> | 2-989 |
| return | size of R3000 data cache. | <i>cacheR3kDsize()</i> | 2-69 |
| cache. return | size of R3000 instruction | <i>cacheR3kIsize()</i> | 2-69 |
| display or set | size of shell history. | <i>h()</i> | 2-246 |
| display or set | size of shell history. | <i>shellHistory()</i> | 2-739 |
| /page block | size (VxVMI Opt.). | <i>vmPageBlockSizeGet()</i> | 2-967 |
| return page | size (VxVMI Opt.). | <i>vmPageSizeGet()</i> | 2-967 |
| set baud rate for | SLIP interface. | <i>slipBaudSet()</i> | 2-756 |
| delete | SLIP interface. | <i>slipDelete()</i> | 2-757 |
| initialize | SLIP interface. | <i>slipInit()</i> | 2-757 |
| driver. Serial Line IP | (SLIP) network interface | <i>if_sl</i> | 1-164 |
| agent. initialize | SLIP packet device for WDB | <i>wdbSlipPktDevInit()</i> | 2-1056 |
| driver and device. publish | sm interface and initialize | <i>smIfAttach()</i> | 2-758 |
| initialize | SMC. | <i>ppc860DevInit()</i> | 2-576 |
| display statistics for | SMC 8013WC elc network/ | <i>elcShow()</i> | 2-161 |
| interface driver. | SMC 8013WC Ethernet network | <i>if_elc</i> | 1-144 |
| network interface driver. | SMC Elite Ultra Ethernet | <i>if_ultra</i> | 1-171 |
| handle | SMC interrupt. | <i>ppc860Int()</i> | 2-576 |
| Motorola MPC800 | SMC UART serial driver. | <i>ppc860Sio</i> | 1-292 |
| interface driver. | SMC Ultra Elite END network | <i>ultraEnd</i> | 1-404 |
| initialize driver and/ publish | sn network interface and | <i>snattach()</i> | 2-779 |
| MIB-II ICMP-group API for | SNMP Agents. | <i>m2lcmpLib</i> | 1-199 |
| MIB-II interface-group API for | SNMP agents. | <i>m2IfLib</i> | 1-200 |
| MIB-II IP-group API for | SNMP agents. | <i>m2IpLib</i> | 1-201 |
| MIB-II API library for | SNMP agents. | <i>m2Lib</i> | 1-203 |
| MIB-II system-group API for | SNMP agents. | <i>m2SysLib</i> | 1-206 |
| MIB-II TCP-group API for | SNMP agents. | <i>m2TcpLib</i> | 1-207 |
| MIB-II UDP-group API for | SNMP agents. | <i>m2UdpLib</i> | 1-209 |
| default IO routines for | SNMP master agent. | <i>masterIoLib</i> | 1-215 |
| create IPC mechanism at | SNMP master agent. | <i>masterIoInit()</i> | 2-396 |
| IPC resources allocated by | SNMP master agent. free | <i>masterIpcFree()</i> | 2-399 |
| free resources allocated for | SNMP master agent. | <i>masterQueCleanup()</i> | 2-401 |
| initialize | SNMP MIB-2 library. | <i>m2Init()</i> | 2-363 |
| default transport routines for | SNMP subagent. | <i>saloLib</i> | 1-318 |
| inform SCSI that hardware | snooping of caches is/ | <i>scsiCacheSnoopDisable()</i> | 2-675 |
| inform SCSI that hardware | snooping of caches is enabled. | <i>scsiCacheSnoopEnable()</i> | 2-675 |
| user data and send it to TCP | socket. create zbuf from | <i>zbufSockBufSend()</i> | 2-1078 |

| | Keyword | Name | Page |
|-------------------------------|--------------------------------------|-----------------------------|--------|
| message and send it to UDP | socket. create zbuf from user | <i>zbufSockBufSendto()</i> | 2-1079 |
| receive data in zbuf from TCP | socket. | <i>zbufSockRecv()</i> | 2-1080 |
| message in zbuf from UDP | socket. receive | <i>zbufSockRecvfrom()</i> | 2-1081 |
| send zbuf data to TCP | socket. | <i>zbufSockSend()</i> | 2-1082 |
| send zbuf message to UDP | socket. | <i>zbufSockSendto()</i> | 2-1083 |
| accept connection from | socket. | <i>accept()</i> | 2-2 |
| enable connections to | socket. | <i>listen()</i> | 2-319 |
| bind name to | socket. | <i>bind()</i> | 2-34 |
| receive data from | socket. | <i>recv()</i> | 2-617 |
| receive message from | socket. | <i>recvfrom()</i> | 2-618 |
| receive message from | socket. | <i>recvmsg()</i> | 2-618 |
| send data to | socket. | <i>send()</i> | 2-725 |
| send message to | socket. | <i>sendmsg()</i> | 2-726 |
| send message to | socket. | <i>sendto()</i> | 2-727 |
| open | socket. | <i>socket()</i> | 2-799 |
| initiate connection to | socket. | <i>connect()</i> | 2-92 |
| try to connect over | socket for specified duration. | <i>connectWithTimeout()</i> | 2-92 |
| zbuf | socket interface library. | <i>zbufSockLib</i> | 1-446 |
| initialize zbuf | socket interface library. | <i>zbufSockLibInit()</i> | 2-1080 |
| generic | socket library. | <i>sockLib</i> | 1-368 |
| get | socket name. | <i>getsockname()</i> | 2-242 |
| get | socket options. | <i>getsockopt()</i> | 2-243 |
| set | socket options. | <i>setsockopt()</i> | 2-732 |
| bind | socket to privileged IP port. | <i>bindresvport()</i> | 2-35 |
| bound to it. open | socket with privileged port | <i>rresvport()</i> | 2-658 |
| Nat. Semi DP83932B | SONIC Ethernet driver. | <i>sn83932End</i> | 1-364 |
| /Semiconductor DP83932B | SONIC Ethernet network driver. | <i>if_sn</i> | 1-167 |
| Interface: I/O DMA library | (SPARC). /L64862 MBus-to-SBus | <i>mmuL64862Lib</i> | 1-232 |
| ROM MMU initialization | (SPARC). | <i>mmuSparcILib</i> | 1-237 |
| window invalid mask register | (SPARC). return contents of | <i>wim()</i> | 2-1064 |
| return contents of y register | (SPARC). | <i>y()</i> | 2-1067 |
| fsr value, symbolically | (SPARC). /meaning of specified | <i>fsrShow()</i> | 2-227 |
| of register i0 (also i1 - i7) | (SPARC). return contents | <i>i0()</i> | 2-253 |
| eight bytes at a time | (SPARC). /buffer to another | <i>bcopyDoubles()</i> | 2-30 |
| of register l0 (also l1 - l7) | (SPARC). return contents | <i>l0()</i> | 2-312 |
| specified eight-byte pattern | (SPARC). fill buffer with | <i>bfillDoubles()</i> | 2-33 |
| I/O MMU DMA data structures | (SPARC). initialize L64862 | <i>mmuL64862DmaInit()</i> | 2-433 |
| initialize MMU for ROM | (SPARC). | <i>mmuSparcRomInit()</i> | 2-434 |
| eight bytes at a time | (SPARC). zero out buffer | <i>bzeroDoubles()</i> | 2-47 |
| of next program counter | (SPARC). return contents | <i>npc()</i> | 2-534 |
| of register o0 (also o1 - o7) | (SPARC). return contents | <i>o0()</i> | 2-541 |
| of processor status register | (SPARC). return contents | <i>psr()</i> | 2-600 |
| psr value, symbolically | (SPARC). /meaning of specified | <i>psrShow()</i> | 2-600 |
| in ASI space for bus error | (SPARC). probe address | <i>vxMemProbeAsi()</i> | 2-974 |
| /of register g0, also g1 - g7 | (SPARC) and g1 - g14 (i960). | <i>g0()</i> | 2-238 |
| buffer manipulation library | SPARC assembly language/ | <i>bALib</i> | 1-27 |
| library. Cypress CY7C604/605 | SPARC cache management | <i>cacheCy604Lib</i> | 1-35 |
| set interrupt level (MC680x0, | SPARC, i960, x86, ARM). | <i>intLevelSet()</i> | 2-279 |
| /lock-out level (MC680x0, | SPARC, i960, x86, ARM). | <i>intLockLevelGet()</i> | 2-282 |
| /lock-out level (MC680x0, | SPARC, i960, x86, ARM). | <i>intLockLevelSet()</i> | 2-282 |

| | Keyword | Name | Page |
|--|---------|----------------------------------|--------|
| /vector table (MC680x0, SPARC, i960, x86, ARM). | | <i>intVecTableWriteProtect()</i> | 2-288 |
| /for C routine (MC680x0, SPARC, i960, x86, MIPS). | | <i>intHandlerCreate()</i> | 2-279 |
| get interrupt vector (MC680x0, SPARC, i960, x86, MIPS). | | <i>intVecGet()</i> | 2-286 |
| CPU vector (trap) (MC680x0, SPARC, i960, x86, MIPS). set | | <i>intVecSet()</i> | 2-286 |
| /(trap) base address (MC680x0, SPARC, i960, x86, MIPS, ARM). | | <i>intVecBaseGet()</i> | 2-284 |
| /(trap) base address (MC680x0, SPARC, i960, x86, MIPS, ARM). | | <i>intVecBaseSet()</i> | 2-285 |
| create and spawn task. | | <i>VXWTask::VXWTask()</i> | 2-1043 |
| spawn task. | | <i>taskSpawn()</i> | 2-884 |
| periodically. spawn task to call function | | <i>period()</i> | 2-567 |
| repeatedly. spawn task to call function | | <i>repeat()</i> | 2-622 |
| parameters. spawn task with default | | <i>sp()</i> | 2-799 |
| snmpQueMonitor(). spawn tMonQue to run | | <i>snmpMonitorSpawn()</i> | 2-787 |
| control structure for MB87030 SPC. create | | <i>mb87030CtrlCreate()</i> | 2-405 |
| control structure for MB87030 SPC. initialize | | <i>mb87030CtrlInit()</i> | 2-406 |
| values of all readable MB87030 SPC registers. display | | <i>mb87030Show()</i> | 2-407 |
| /of failed attempts to take spin-lock (VxMP Opt.). | | <i>smObjTimeoutLogEnable()</i> | 2-778 |
| zbufs. split zbuf into two separate | | <i>zbufSplit()</i> | 2-1084 |
| spy CPU activity library. | | <i>spyLib</i> | 1-369 |
| stop spying and reporting. | | <i>spyStop()</i> | 2-804 |
| compute non-negative square root (ANSI). | | <i>sqrt()</i> | 2-804 |
| compute non-negative square root (ANSI). | | <i>sqrtf()</i> | 2-805 |
| PCMCIA SRAM device driver. | | <i>sramDrv</i> | 1-370 |
| install PCMCIA SRAM memory driver. | | <i>sramDrv()</i> | 2-807 |
| initialize task with specified stack. | | <i>VXWTask::VXWTask()</i> | 2-1045 |
| routine for TCP/IP network stack. generic attach | | <i>ipAttach()</i> | 2-299 |
| routine for TCP/IP network stack. generic detach | | <i>ipDetach()</i> | 2-299 |
| initialize task with stack at specified address. | | <i>taskInit()</i> | 2-873 |
| show network stack data pool statistics. | | <i>netStackDataPoolShow()</i> | 2-519 |
| show network stack system pool statistics. | | <i>netStackSysPoolShow()</i> | 2-520 |
| display stack trace of task. | | <i>tt()</i> | 2-925 |
| print summary of each task's stack usage. | | <i>checkStack()</i> | 2-84 |
| for standard output or standard error. /buffering | | <i>setlinebuf()</i> | 2-731 |
| write formatted string to standard error stream. | | <i>printErr()</i> | 2-590 |
| return next character from standard input stream (ANSI). | | <i>getchar()</i> | 2-239 |
| read characters from standard input stream (ANSI). | | <i>gets()</i> | 2-242 |
| /and convert characters from standard input stream (ANSI). | | <i>scanf()</i> | 2-666 |
| get fd for global standard input/output/error. | | <i>ioGlobalStdGet()</i> | 2-290 |
| set fd for global standard input/output/error. | | <i>ioGlobalStdSet()</i> | 2-290 |
| get fd for task standard input/output/error. | | <i>ioTaskStdGet()</i> | 2-298 |
| set fd for task standard input/output/error. | | <i>ioTaskStdSet()</i> | 2-298 |
| FILE of current task. return standard input/output/error | | <i>stdioFp()</i> | 2-816 |
| initialize standard I/O show facility. | | <i>stdioShowInit()</i> | 2-817 |
| initialize standard I/O support. | | <i>stdioInit()</i> | 2-816 |
| /with variable argument list to standard output (ANSI). | | <i>vprintf()</i> | 2-971 |
| error. set line buffering for standard output or standard | | <i>setlinebuf()</i> | 2-731 |
| write formatted string to standard output stream (ANSI). | | <i>printf()</i> | 2-591 |
| write character to standard output stream (ANSI). | | <i>putchar()</i> | 2-603 |
| write string to standard output stream (ANSI). | | <i>puts()</i> | 2-604 |
| ANSI stdarg documentation. | | <i>ansiStdarg</i> | 1-15 |
| ANSI stdio documentation. | | <i>ansiStdio</i> | 1-16 |

| | Keyword | Name | Page |
|--------------------------------|-------------------------------------|--------------------------------|--------|
| | ANSI stdlib documentation. | ansiStdlib | 1-20 |
| copy in (or stdin) to out (or | stdout). | <i>copy()</i> | 2-93 |
| word (32-bit integer) from | stream. read next | <i>getw()</i> | 2-244 |
| string to standard error | stream. write formatted | <i>printErr()</i> | 2-590 |
| write word (32-bit integer) to | stream. | <i>putw()</i> | 2-605 |
| specify buffering for | stream. | <i>setbuffer()</i> | 2-729 |
| close | stream (ANSI). | <i>fclose()</i> | 2-192 |
| test end-of-file indicator for | stream (ANSI). | <i>feof()</i> | 2-198 |
| flush | stream (ANSI). | <i>fflush()</i> | 2-199 |
| return next character from | stream (ANSI). | <i>fgetc()</i> | 2-200 |
| of file position indicator for | stream (ANSI). /current value | <i>fgetpos()</i> | 2-200 |
| number of characters from | stream (ANSI). read specified | <i>fgets()</i> | 2-201 |
| write formatted string to | stream (ANSI). | <i>fprintf()</i> | 2-215 |
| write character to | stream (ANSI). | <i>fputc()</i> | 2-219 |
| write string to | stream (ANSI). | <i>fputs()</i> | 2-219 |
| and convert characters from | stream (ANSI). read | <i>fscanf()</i> | 2-222 |
| file position indicator for | stream (ANSI). set | <i>fseek()</i> | 2-226 |
| file position indicator for | stream (ANSI). set | <i>fsetpos()</i> | 2-227 |
| of file position indicator for | stream (ANSI). /current value | <i>ftell()</i> | 2-229 |
| return next character from | stream (ANSI). | <i>getc()</i> | 2-239 |
| character from standard input | stream (ANSI). return next | <i>getchar()</i> | 2-239 |
| characters from standard input | stream (ANSI). read | <i>gets()</i> | 2-242 |
| string to standard output | stream (ANSI). /formatted | <i>printf()</i> | 2-591 |
| write character to | stream (ANSI). | <i>putc()</i> | 2-603 |
| character to standard output | stream (ANSI). write | <i>putchar()</i> | 2-603 |
| string to standard output | stream (ANSI). write | <i>puts()</i> | 2-604 |
| characters from standard input | stream (ANSI). /and convert | <i>scanf()</i> | 2-666 |
| specify buffering for | stream (ANSI). | <i>setbuf()</i> | 2-728 |
| specify buffering for | stream (ANSI). | <i>setobuf()</i> | 2-737 |
| and error flags for | stream (ANSI). /end-of-file | <i>clearerr()</i> | 2-88 |
| push character back into input | stream (ANSI). | <i>ungetc()</i> | 2-940 |
| write formatted string to | stream (ANSI). | <i>vfprintf()</i> | 2-956 |
| transfer file via TFTP using | stream interface. | <i>tftpXfer()</i> | 2-907 |
| return fd for | stream (POSIX). | <i>fileno()</i> | 2-201 |
| copy from/to specified | streams. | <i>copyStreams()</i> | 2-94 |
| get task status as | string. | <i>VXWTask::statusString()</i> | 2-1038 |
| parse initialization | string. | <i>el3c90xInitParse()</i> | 2-159 |
| parse init | string. | <i>elt3c509Parse()</i> | 2-162 |
| occurrence of character in | string. find first | <i>index()</i> | 2-268 |
| return kernel revision | string. | <i>kernelVersion()</i> | 2-311 |
| parse initialization | string. | <i>ln97xInitParse()</i> | 2-321 |
| parse initialization | string. | <i>ln7990InitParse()</i> | 2-324 |
| change login | string. | <i>loginStringSet()</i> | 2-341 |
| parse initialization | string. | <i>mb86960InitParse()</i> | 2-404 |
| parse init | string. | <i>mbcParse()</i> | 2-411 |
| parse init | string. | <i>ne2000Parse()</i> | 2-502 |
| parse initialization | string. | <i>nicEvbInitParse()</i> | 2-533 |
| parse init | string. | <i>ntParse()</i> | 2-539 |
| occurrence of character in | string. find last | <i>rindex()</i> | 2-632 |
| get task's status as | string. | <i>taskStatusString()</i> | 2-886 |

| | Keyword | Name | Page |
|--|--------------------------------|---|--------|
| | parse init | string. <i>ultraParse()</i> | 2-938 |
| | convert time in seconds into | string (ANSI). <i>ctime()</i> | 2-106 |
| | convert broken-down time into | string (ANSI). <i>asctime()</i> | 2-18 |
| | convert characters from ASCII | string (ANSI). read and <i>sscanf()</i> | 2-808 |
| | occurrence of character in | string (ANSI). find first <i>strchr()</i> | 2-818 |
| | map error number to error | string (ANSI). <i>strerror()</i> | 2-821 |
| | time into formatted | string (ANSI). /broken-down <i>strftime()</i> | 2-822 |
| | determine length of | string (ANSI). <i>strlen()</i> | 2-823 |
| | occurrence of character in | string (ANSI). find last <i>strrchr()</i> | 2-826 |
| | occurrence of substring in | string (ANSI). find first <i>strstr()</i> | 2-827 |
| | ANSI | string documentation. <i>ansiString</i> | 1-21 |
| | argument list to buffer/ write | string formatted with variable <i>vsprintf()</i> | 2-972 |
| | argument list to fd. write | string formatted with variable <i>vfdprintf()</i> | 2-956 |
| | argument list to/ write | string formatted with variable <i>vprintf()</i> | 2-971 |
| | read | string from file. <i>fileReadString()</i> | 2-203 |
| | database (VxMP/ define name | string in shared-memory name <i>VXWSmName::nameSet()</i> | 2-1020 |
| | break down | string into tokens (ANSI). <i>strtok()</i> | 2-828 |
| | (POSIX). break down | string into tokens (reentrant) <i>strtok_r()</i> | 2-829 |
| | character from given/ return | string length up to first <i>strcspn()</i> | 2-820 |
| | character not in given/ return | string length up to first <i>strspn()</i> | 2-826 |
| | find device using | string name. <i>endFindByName()</i> | 2-166 |
| | set/ find first occurrence in | string of character from given <i>strpbrk()</i> | 2-825 |
| | convert time in seconds into | string (POSIX). <i>ctime_r()</i> | 2-107 |
| | convert broken-down time into | string (POSIX). <i>asctime_r()</i> | 2-18 |
| | map error number to error | string (POSIX). <i>strerror_r()</i> | 2-821 |
| | Internet network number from | string to address. convert <i>inet_network()</i> | 2-272 |
| | concatenate one | string to another (ANSI). <i>strcat()</i> | 2-818 |
| | copy one | string to another (ANSI). <i>strcpy()</i> | 2-820 |
| | /characters from one | string to another (ANSI). <i>strncat()</i> | 2-824 |
| | copy characters from one | string to another (ANSI). <i>strncpy()</i> | 2-825 |
| | write formatted | string to buffer (ANSI). <i>sprintf()</i> | 2-800 |
| | (Unimplemented) (ANSI). pass | string to command processor <i>system()</i> | 2-859 |
| | convert | string to double (ANSI). <i>atof()</i> | 2-26 |
| | convert initial portion of | string to double (ANSI). <i>strtod()</i> | 2-827 |
| | write formatted | string to fd. <i>fdprintf()</i> | 2-195 |
| | post user event | string to host tools.. <i>wdbUserEvtPost()</i> | 2-1059 |
| | convert | string to int (ANSI). <i>atoi()</i> | 2-26 |
| | convert | string to long (ANSI). <i>atol()</i> | 2-27 |
| | convert | string to long integer (ANSI). <i>strtol()</i> | 2-830 |
| | stream. write formatted | string to standard error <i>printErr()</i> | 2-590 |
| | stream/ write formatted | string to standard output <i>printf()</i> | 2-591 |
| | stream (ANSI). write | string to standard output <i>puts()</i> | 2-604 |
| | write formatted | string to stream (ANSI). <i>fprintf()</i> | 2-215 |
| | write | string to stream (ANSI). <i>fputs()</i> | 2-219 |
| | write formatted | string to stream (ANSI). <i>vfprintf()</i> | 2-956 |
| | integer (ANSI). convert | string to unsigned long <i>strtoul()</i> | 2-831 |
| | first n characters of two | strings (ANSI). compare <i>strncmp()</i> | 2-824 |
| | LC_COLLATE/ compare two | strings as appropriate to <i>strcoll()</i> | 2-819 |
| | (ANSI). compare two | strings lexicographically <i>strcmp()</i> | 2-819 |
| | define | subnet for network interface. <i>ifMaskSet()</i> | 2-264 |

| | Keyword | Name | Page |
|-------------------------------|--------------------------------------|--|--------|
| interface. get | subnet mask for network | <i>ifMaskGet()</i> | 2-263 |
| library. TI TMS390 | SuperSPARC cache management | <i>cacheTITms390Lib</i> | 1-50 |
| time interval elapses/ | suspend current task until | <i>nanosleep()</i> | 2-484 |
| | suspend system.. | <i>wdbSystemSuspend()</i> | 2-1056 |
| | suspend task. | <i>VXWTask::suspend()</i> | 2-1039 |
| | suspend task. | <i>taskSuspend()</i> | 2-887 |
| | suspend task. | <i>ts()</i> | 2-924 |
| signal (POSIX). | suspend task until delivery of | <i>pause()</i> | 2-550 |
| signal (POSIX). | suspend task until delivery of | <i>sigsuspend()</i> | 2-749 |
| check if task is | suspended. | <i>VXWTask::isSuspended()</i> | 2-1030 |
| check if task is | suspended. | <i>taskIsSuspended()</i> | 2-875 |
| | swap buffers. | <i>bswap()</i> | 2-46 |
| | swap bytes. | <i>swab()</i> | 2-833 |
| are not necessarily aligned. | swap bytes with buffers that | <i>uswab()</i> | 2-952 |
| to be called at every task | switch. add routine | <i>taskSwitchHookAdd()</i> | 2-888 |
| delete previously added task | switch routine. | <i>taskSwitchHookDelete()</i> | 2-889 |
| show list of task | switch routines. | <i>taskSwitchHookShow()</i> | 2-889 |
| look up | symbol by name. | <i>VXWSymTab::findByName()</i> | 2-1023 |
| look up | symbol by name. | <i>symFindByName()</i> | 2-835 |
| look up | symbol by name and type. | <i>VXWSymTab::findByNameAndType()</i> | 2-1023 |
| look up | symbol by name and type. | <i>symFindByNameAndType()</i> | 2-836 |
| look up | symbol by value. | <i>VXWSymTab::findByValue()</i> | 2-1024 |
| look up | symbol by value. | <i>symFindByValue()</i> | 2-836 |
| look up | symbol by value and type. | <i>VXWSymTab::findByValueAndType()</i> | 2-1024 |
| look up | symbol by value and type. | <i>symFindByValueAndType()</i> | 2-837 |
| remove | symbol from symbol table. | <i>VXWSymTab::remove()</i> | 2-1025 |
| remove | symbol from symbol table. | <i>symRemove()</i> | 2-838 |
| to examine each entry in | symbol table. call routine | <i>VXWSymTab::each()</i> | 2-1022 |
| remove symbol from | symbol table. | <i>VXWSymTab::remove()</i> | 2-1025 |
| create | symbol table. | <i>VXWSymTab::VXWSymTab()</i> | 2-1025 |
| delete | symbol table. | <i>VXWSymTab::~~VXWSymTab()</i> | 2-1026 |
| to examine each entry in | symbol table. call routine | <i>symEach()</i> | 2-834 |
| remove symbol from | symbol table. | <i>symRemove()</i> | 2-838 |
| create | symbol table. | <i>symTblCreate()</i> | 2-839 |
| delete | symbol table. | <i>symTblDelete()</i> | 2-840 |
| | symbol table class. | <i>VXWSymTab</i> | 1-426 |
| create and add symbol to | symbol table, including group/ | <i>VXWSymTab::add()</i> | 2-1021 |
| create and add symbol to | symbol table, including group/ | <i>symAdd()</i> | 2-834 |
| initialize | symbol table library. | <i>symLibInit()</i> | 2-838 |
| library. | symbol table subroutine | <i>symLib</i> | 1-374 |
| host/target | symbol table synchronization. | <i>symSyncLib</i> | 1-376 |
| initialize host/target | symbol table synchronization. | <i>symSyncLibInit()</i> | 2-839 |
| including/ create and add | symbol to symbol table, | <i>VXWSymTab::add()</i> | 2-1021 |
| including/ create and add | symbol to symbol table, | <i>symAdd()</i> | 2-834 |
| list | symbols. | <i>lkup()</i> | 2-320 |
| specified value. list | symbols whose values are near | <i>lkAddr()</i> | 2-319 |
| bypass RIP and kernel routing | tables. add hook to | <i>ripLeakHookAdd()</i> | 2-639 |
| host and network routing | tables. display | <i>routeShow()</i> | 2-656 |
| semaphore is available. | take release 4.x semaphore, if | <i>semClear()</i> | 2-710 |
| | take semaphore. | <i>VXWSem::take()</i> | 2-1017 |

| | Keyword | Name | Page |
|--------------------------------|-----------------------------|--------------------------------|--------|
| | take semaphore. | <i>semTake()</i> | 2-718 |
| /logging of failed attempts to | take spin-lock (VxMP Opt.). | <i>smObjTimeoutLogEnable()</i> | 2-778 |
| compute arc | tangent (ANSI). | <i>atan()</i> | 2-22 |
| compute arc | tangent (ANSI). | <i>atanf()</i> | 2-23 |
| compute | tangent (ANSI). | <i>tan()</i> | 2-860 |
| compute | tangent (ANSI). | <i>tanf()</i> | 2-861 |
| compute hyperbolic | tangent (ANSI). | <i>tanh()</i> | 2-861 |
| compute hyperbolic | tangent (ANSI). | <i>tanhf()</i> | 2-862 |
| compute arc | tangent of y/x (ANSI). | <i>atan2()</i> | 2-22 |
| compute arc | tangent of y/x (ANSI). | <i>atan2f()</i> | 2-23 |
| exit | task (ANSI). | <i>exit()</i> | 2-190 |
| activate | task. | <i>VXWTask::activate()</i> | 2-1027 |
| get information about | task. | <i>VXWTask::info()</i> | 2-1029 |
| send signal to | task. | <i>VXWTask::kill()</i> | 2-1030 |
| examine priority of | task. | <i>VXWTask::priority()</i> | 2-1032 |
| change priority of | task. | <i>VXWTask::priority()</i> | 2-1033 |
| restart | task. | <i>VXWTask::restart()</i> | 2-1034 |
| resume | task. | <i>VXWTask::resume()</i> | 2-1035 |
| send queued signal to | task. | <i>VXWTask::sigqueue()</i> | 2-1037 |
| suspend | task. | <i>VXWTask::suspend()</i> | 2-1039 |
| add task variable to | task. | <i>VXWTask::varAdd()</i> | 2-1040 |
| remove task variable from | task. | <i>VXWTask::varDelete()</i> | 2-1041 |
| create and spawn | task. | <i>VXWTask::VXWTask()</i> | 2-1043 |
| delete | task. | <i>VXWTask::~VXWTask()</i> | 2-1046 |
| display environment for | task. | <i>envShow()</i> | 2-171 |
| error status value of calling | task. get | <i>errnoGet()</i> | 2-172 |
| status value of specified | task. get error | <i>errnoOfTaskGet()</i> | 2-172 |
| status value of specified | task. set error | <i>errnoOfTaskSet()</i> | 2-173 |
| error status value of calling | task. set | <i>errnoSet()</i> | 2-173 |
| floating-point registers of | task. set | <i>fppTaskRegsSet()</i> | 2-214 |
| terminate FTP server | task. | <i>ftpdDelete()</i> | 2-232 |
| initialize FTP server | task. | <i>ftpdInit()</i> | 2-232 |
| message-logging support | task. | <i>logTask()</i> | 2-345 |
| send signal to caller's | task. | <i>raise()</i> | 2-607 |
| single-step | task. | <i>s()</i> | 2-662 |
| send queued signal to | task. | <i>sigqueue()</i> | 2-748 |
| FILE of current | task. /input/output/error | <i>stdioFp()</i> | 2-816 |
| delete | task. | <i>taskDelete()</i> | 2-867 |
| get task ID of running | task. | <i>taskIdSelf()</i> | 2-872 |
| verify existence of | task. | <i>taskIdVerify()</i> | 2-872 |
| get information about | task. | <i>taskInfoGet()</i> | 2-873 |
| examine priority of | task. | <i>taskPriorityGet()</i> | 2-878 |
| change priority of | task. | <i>taskPrioritySet()</i> | 2-878 |
| restart | task. | <i>taskRestart()</i> | 2-881 |
| resume | task. | <i>taskResume()</i> | 2-881 |
| spawn | task. | <i>taskSpawn()</i> | 2-884 |
| suspend | task. | <i>taskSuspend()</i> | 2-887 |
| add task variable to | task. | <i>taskVarAdd()</i> | 2-891 |
| remove task variable from | task. | <i>taskVarDelete()</i> | 2-892 |
| get list of task variables of | task. | <i>taskVarInfo()</i> | 2-893 |

| | Keyword | Name | Page |
|-------------------------------|-----------------------------------|-------------------------------|--------|
| | delete task. | <i>td()</i> | 2-898 |
| initialize TFTP server | task. | <i>tftpdInit()</i> | 2-901 |
| TFTP server daemon | task. | <i>tftpdTask()</i> | 2-901 |
| resume | task. | <i>tr()</i> | 2-922 |
| suspend | task. | <i>ts()</i> | 2-924 |
| display stack trace of | task. | <i>tt()</i> | 2-925 |
| start collecting | task activity data. | <i>spyClkStart()</i> | 2-801 |
| stop collecting | task activity data. | <i>spyClkStop()</i> | 2-802 |
| display | task activity data. | <i>spyReport()</i> | 2-803 |
| begin periodic | task activity reports. | <i>spy()</i> | 2-801 |
| run periodic | task activity reports. | <i>spyTask()</i> | 2-804 |
| | task class. | <i>VXWTask</i> | 1-427 |
| state is in interrupt or | task context. /if current | <i>intContext()</i> | 2-276 |
| get | task control block. | <i>VXWTask::tcb()</i> | 2-1039 |
| ID. get | task control block for task | <i>taskTcb()</i> | 2-889 |
| package. initialize | task CPU utilization tool | <i>spyLibInit()</i> | 2-803 |
| routine to be called at every | task create. add | <i>taskCreateHookAdd()</i> | 2-865 |
| delete previously added | task create routine. | <i>taskCreateHookDelete()</i> | 2-866 |
| show list of | task create routines. | <i>taskCreateHookShow()</i> | 2-866 |
| routine to be called at every | task delete. add | <i>taskDeleteHookAdd()</i> | 2-868 |
| delete previously added | task delete routine. | <i>taskDeleteHookDelete()</i> | 2-869 |
| show list of | task delete routines. | <i>taskDeleteHookShow()</i> | 2-869 |
| network | task entry point. | <i>netTask()</i> | 2-520 |
| delay | task from executing. | <i>taskDelay()</i> | 2-866 |
| initialize | task hook facilities. | <i>taskHookInit()</i> | 2-870 |
| | task hook library. | <i>taskHookLib</i> | 1-384 |
| initialize | task hook show facility. | <i>taskHookShowInit()</i> | 2-870 |
| | task hook show routines. | <i>taskHookShow</i> | 1-386 |
| reveal | task ID. | <i>VXWTask::id()</i> | 2-1029 |
| get name associated with | task ID. | <i>VXWTask::name()</i> | 2-1031 |
| set default | task ID. | <i>taskIdDefault()</i> | 2-871 |
| get name associated with | task ID. | <i>taskName()</i> | 2-876 |
| get task control block for | task ID. | <i>taskTcb()</i> | 2-889 |
| name. look up | task ID associated with task | <i>taskNameToId()</i> | 2-876 |
| get | task ID of running task. | <i>taskIdSelf()</i> | 2-872 |
| get list of active | task IDs. | <i>taskIdListGet()</i> | 2-871 |
| semaphore. get list of | task IDs that are blocked on | <i>VXWSem::info()</i> | 2-1016 |
| semaphore. get list of | task IDs that are blocked on | <i>semInfo()</i> | 2-713 |
| display | task information from TCBS. | <i>VXWTask::show()</i> | 2-1036 |
| display | task information from TCBS. | <i>taskShow()</i> | 2-882 |
| | task information library. | <i>taskInfo</i> | 1-386 |
| check if | task is ready to run. | <i>VXWTask::isReady()</i> | 2-1030 |
| check if | task is ready to run. | <i>taskIsReady()</i> | 2-874 |
| check if | task is suspended. | <i>VXWTask::isSuspended()</i> | 2-1030 |
| check if | task is suspended. | <i>taskIsSuspended()</i> | 2-875 |
| | task management library. | <i>taskLib</i> | 1-387 |
| architecture-specific | task management routines. | <i>taskArchLib</i> | 1-384 |
| display | task monitoring help menu. | <i>spyHelp()</i> | 2-802 |
| up task ID associated with | task name. look | <i>taskNameToId()</i> | 2-876 |
| initialize | task object. | <i>VXWTask::VXWTask()</i> | 2-1043 |

| | Keyword | Name | Page |
|--|--|--------------------------------|--------|
| | examine task options. | VXWTask::options() | 2-1031 |
| | change task options. | VXWTask::options() | 2-1032 |
| | examine task options. | taskOptionsGet() | 2-877 |
| | change task options. | taskOptionsSet() | 2-877 |
| | wake up task pending in <i>select()</i> | selWakeup() | 2-705 |
| | unblock every task pending on semaphore. | VXWSem::flush() | 2-1014 |
| | unblock every task pending on semaphore. | semFlush() | 2-712 |
| | send signal to task (POSIX). | kill() | 2-311 |
| | parameters for specified task (POSIX). get scheduling | sched_getparam() | 2-668 |
| | display contents of task registers. | VXWTask::show() | 2-1035 |
| | get task registers from TCB. | VXWTask::registers() | 2-1034 |
| | disable task rescheduling. | taskLock() | 2-875 |
| | enable task rescheduling. | taskUnlock() | 2-890 |
| | make calling task safe from deletion. | taskSafe() | 2-882 |
| | initialize task show routine facility. | taskShowInit() | 2-883 |
| | task show routines. | taskShow | 1-389 |
| | get fd for task standard/ | ioTaskStdGet() | 2-298 |
| | set fd for task standard/ | ioTaskStdSet() | 2-298 |
| | get task status as string. | VXWTask::statusString() | 2-1038 |
| | MIPS, i386/i486). set task status register (MC680x0, | VXWTask::SRSet() | 2-1037 |
| | MIPS, i386/i486). set task status register (MC680x0, | taskSRSet() | 2-886 |
| | initialize default task status register (MIPS). | taskSRInit() | 2-885 |
| | routine to be called at every task switch. add | taskSwitchHookAdd() | 2-888 |
| | delete previously added task switch routine. | taskSwitchHookDelete() | 2-889 |
| | show list of task switch routines. | taskSwitchHookShow() | 2-889 |
| | floating-point registers from task TCB. get | fppTaskRegsGet() | 2-213 |
| | initialized. activate task that has been | taskActivate() | 2-865 |
| | on queue (POSIX). notify task that message is available | mq_notify() | 2-449 |
| | periodically. spawn task to call function | period() | 2-567 |
| | repeatedly. spawn task to call function | repeat() | 2-622 |
| | make calling task unsafe from deletion. | taskUnsafe() | 2-890 |
| | (POSIX). suspend task until delivery of signal | pause() | 2-550 |
| | (POSIX). suspend task until delivery of signal | sigsuspend() | 2-749 |
| | elapses/ suspend current task until time interval | nanosleep() | 2-484 |
| | get value of task variable. | VXWTask::varGet() | 2-1041 |
| | set value of task variable. | VXWTask::varSet() | 2-1042 |
| | get value of task variable. | taskVarGet() | 2-893 |
| | set value of task variable. | taskVarSet() | 2-895 |
| | remove task variable from task. | VXWTask::varDelete() | 2-1041 |
| | remove task variable from task. | taskVarDelete() | 2-892 |
| | add task variable to task. | VXWTask::varAdd() | 2-1040 |
| | add task variable to task. | taskVarAdd() | 2-891 |
| | get list of task variables. | VXWTask::varInfo() | 2-1042 |
| | initialize task variables facility. | taskVarInit() | 2-894 |
| | get list of task variables of task. | taskVarInfo() | 2-893 |
| | library. task variables support | taskVarLib | 1-390 |
| | spawn task with default parameters. | sp() | 2-799 |
| | initialize task with specified stack. | VXWTask::VXWTask() | 2-1045 |
| | address. initialize task with stack at specified | taskInit() | 2-873 |
| | delete task without restriction. | VXWTask::deleteForce() | 2-1027 |

| | Keyword | Name | Page |
|-------------------------------|--|-----------------------------|--------|
| | delete task without restriction. | <i>taskDeleteForce()</i> | 2-868 |
| | handle task-level exceptions. | <i>excTask()</i> | 2-188 |
| | handle task-level PCMCIA events. | <i>pcmciaid()</i> | 2-555 |
| | device. do task-level read for tty | <i>tyRead()</i> | 2-932 |
| | device. do task-level write for tty | <i>tyWrite()</i> | 2-933 |
| | initialize task's access to RPC package. | <i>rpcTaskInit()</i> | 2-658 |
| registers. print contents of | task's floating-point | <i>fppTaskRegsShow()</i> | 2-214 |
| | set task's priority (POSIX). | <i>sched_setparam()</i> | 2-670 |
| | set task's registers. | <i>VXWTask::registers()</i> | 2-1033 |
| | set task's registers. | <i>taskRegsSet()</i> | 2-879 |
| display contents of | task's registers. | <i>taskRegsShow()</i> | 2-880 |
| | get task's registers from TCB. | <i>taskRegsGet()</i> | 2-879 |
| print summary of each | task's stack usage. | <i>checkStack()</i> | 2-84 |
| | get task's status as string. | <i>taskStatusString()</i> | 2-886 |
| print summary of each | task's TCB. | <i>i()</i> | 2-252 |
| complete information from | task's TCB. print | <i>ti()</i> | 2-908 |
| get task registers from | TCB. | <i>VXWTask::registers()</i> | 2-1034 |
| registers from task | TCB. get floating-point | <i>fppTaskRegsGet()</i> | 2-213 |
| print summary of each task's | TCB. | <i>i()</i> | 2-252 |
| get task's registers from | TCB. | <i>taskRegsGet()</i> | 2-879 |
| information from task's | TCB. print complete | <i>ti()</i> | 2-908 |
| initialize | TCIC chip. | <i>tcicInit()</i> | 2-895 |
| show all configurations of | TCIC chip. | <i>tcicShow()</i> | 2-896 |
| chip driver. Databook | TCIC/2 PCMCIA host bus adaptor | <i>tcic</i> | 1-390 |
| chip show library. Databook | TCIC/2 PCMCIA host bus adaptor | <i>tcicShow</i> | 1-391 |
| | get MIB-II TCP connection table entry. | <i>m2TcpConnEntryGet()</i> | 2-381 |
| | state. set TCP connection to closed | <i>m2TcpConnEntrySet()</i> | 2-381 |
| all resources used to access | TCP group. delete | <i>m2TcpDelete()</i> | 2-382 |
| routines. | TCP information display | <i>tcpShow</i> | 1-391 |
| debugging information for | TCP protocol. display | <i>tcpDebugShow()</i> | 2-896 |
| display all statistics for | TCP protocol. | <i>tcpstatShow()</i> | 2-897 |
| initialize | TCP show routines. | <i>tcpShowInit()</i> | 2-897 |
| from user data and send it to | TCP socket. create zbuf | <i>zbufSockBufSend()</i> | 2-1078 |
| receive data in zbuf from | TCP socket. | <i>zbufSockRecv()</i> | 2-1080 |
| send zbuf data to | TCP socket. | <i>zbufSockSend()</i> | 2-1082 |
| VxWorks | telnet daemon. | <i>telnetd()</i> | 2-898 |
| initialize | telnet daemon. | <i>telnetInit()</i> | 2-899 |
| | telnet server library. | <i>telnetLib</i> | 1-392 |
| bus. | test and set location across | <i>sysBusTas()</i> | 2-845 |
| C-callable atomic | test-and-set primitive. | <i>vxTas()</i> | 2-977 |
| SCSI device. issue | TEST_UNIT_READY command to | <i>scsiTestUnitRdy()</i> | 2-700 |
| write-protect | text segment (VxVMI Opt.). | <i>vmTextProtect()</i> | 2-970 |
| transfer file via | TFTP. | <i>tftpCopy()</i> | 2-899 |
| | send TFTP message to remote system. | <i>tftpSend()</i> | 2-906 |
| | set TFTP server address. | <i>tftpPeerSet()</i> | 2-904 |
| | TFTP server daemon task. | <i>tftpdTask()</i> | 2-901 |
| initialize | TFTP server task. | <i>tftpdInit()</i> | 2-901 |
| initialize | TFTP session. | <i>tftpInit()</i> | 2-903 |
| quit | TFTP session. | <i>tftpQuit()</i> | 2-905 |
| get | TFTP status information. | <i>tftpInfoShow()</i> | 2-903 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|--------------------------------|--------|
| set | TFTP transfer mode. | <i>tftpModeSet()</i> | 2-904 |
| transfer file via | TFTP using stream interface. | <i>tftpXfer()</i> | 2-907 |
| completed execution of client | thread. successfully | <i>aic7880ScbCompleted()</i> | 2-7 |
| perform generic SCSI | thread initialization. | <i>scsiThreadInit()</i> | 2-700 |
| send event to | thread state machine. | <i>scsiMgrThreadEvent()</i> | 2-683 |
| get value of kernel's | tick counter. | <i>tickGet()</i> | 2-909 |
| set value of kernel's | tick counter. | <i>tickSet()</i> | 2-910 |
| clock | tick support library. | <i>tickLib</i> | 1-395 |
| announce clock | tick to kernel. | <i>tickAnnounce()</i> | 2-909 |
| network with DHCP at boot | time. initialize | <i>dhcpcBootBind()</i> | 2-116 |
| set dosFs file system | time. | <i>dosFsTimeSet()</i> | 2-146 |
| time into UTC broken-down | time (ANSI). convert calendar | <i>gmtime()</i> | 2-245 |
| calendar time into broken-down | time (ANSI). convert | <i>localtime()</i> | 2-333 |
| broken-down time into calendar | time (ANSI). convert | <i>mktime()</i> | 2-432 |
| determine current calendar | time (ANSI). | <i>time()</i> | 2-910 |
| reload value/ get remaining | time before expiration and | <i>timer_gettime()</i> | 2-914 |
| ANSI | time documentation. | <i>ansiTime</i> | 1-22 |
| retrieve current | time from remote source. | <i>sntpTimeGet()</i> | 2-795 |
| (ANSI). convert | time in seconds into string | <i>ctime()</i> | 2-106 |
| (POSIX). convert | time in seconds into string | <i>ctime_r()</i> | 2-107 |
| determine processor | time in use (ANSI). | <i>clock()</i> | 2-88 |
| suspend current task until | time interval elapses (POSIX). | <i>nanosleep()</i> | 2-484 |
| (ANSI). convert calendar | time into broken-down time | <i>localtime()</i> | 2-333 |
| (POSIX). convert calendar | time into broken-down time | <i>gmtime_r()</i> | 2-245 |
| (POSIX). convert calendar | time into broken-down time | <i>localtime_r()</i> | 2-333 |
| (ANSI). convert broken-down | time into calendar time | <i>mktime()</i> | 2-432 |
| (ANSI). convert broken-down | time into formatted string | <i>strftime()</i> | 2-822 |
| convert broken-down | time into string (ANSI). | <i>asctime()</i> | 2-18 |
| convert broken-down | time into string (POSIX). | <i>asctime_r()</i> | 2-18 |
| (ANSI). convert calendar | time into UTC broken-down time | <i>gmtime()</i> | 2-245 |
| get current | time of clock (POSIX). | <i>clock_gettime()</i> | 2-89 |
| update | time on file. | <i>utime()</i> | 2-953 |
| calendar time into broken-down | time (POSIX). convert | <i>gmtime_r()</i> | 2-245 |
| calendar time into broken-down | time (POSIX). convert | <i>localtime_r()</i> | 2-333 |
| set clock to specified | time (POSIX). | <i>clock_settime()</i> | 2-90 |
| library. Simple Network | Time Protocol (SNTP) client | <i>sntpLib</i> | 1-366 |
| library. Simple Network | Time Protocol (SNTP) server | <i>sntpsLib</i> | 1-367 |
| function or group of/ | time repeated executions of | <i>timexN()</i> | 2-918 |
| function or functions. | time single execution of | <i>timex()</i> | 2-915 |
| get current | time slice (POSIX). | <i>sched_rr_get_interval()</i> | 2-669 |
| arm timer (POSIX). set | time until next expiration and | <i>timer_settime()</i> | 2-914 |
| list of function calls to be | timed. clear | <i>timexClear()</i> | 2-916 |
| specify functions to be | timed. | <i>timexFunc()</i> | 2-916 |
| list of function calls to be | timed. display | <i>timexShow()</i> | 2-920 |
| start watchdog | timer. | <i>VXWwd::start()</i> | 2-1047 |
| construct watchdog | timer. | <i>VXWwd::VXWwd()</i> | 2-1047 |
| construct watchdog | timer. | <i>VXWwd::VXWwd()</i> | 2-1048 |
| destroy watchdog | timer. | <i>VXWwd::~~VXWwd()</i> | 2-1048 |
| create watchdog | timer. | <i>wdCreate()</i> | 2-1061 |
| delete watchdog | timer. | <i>wdDelete()</i> | 2-1061 |

| | Keyword | Name | Page |
|--|-------------------------------|------------------------------------|---------------------------------|
| | start watchdog | timer. | <i>wdStart()</i> 2-1063 |
| | cancel | timer. | <i>timer_cancel()</i> 2-911 |
| | watchdog | timer class. | <i>VXWwd</i> 1-429 |
| | (POSIX). return | timer expiration overrun | <i>timer_getoverrun()</i> 2-913 |
| | specified function when | timer expires. execute | <i>envoy_call_timer()</i> 2-169 |
| | execution | timer facilities. | <i>timexLib</i> 1-396 |
| | display synopsis of execution | timer facilities. | <i>timexHelp()</i> 2-917 |
| | watchdog | timer library. | <i>wdLib</i> 1-440 |
| | include execution | timer library. | <i>timexInit()</i> 2-918 |
| | | timer library (POSIX). | <i>timerLib</i> 1-395 |
| | remove previously created | timer (POSIX). | <i>timer_delete()</i> 2-913 |
| | until next expiration and arm | timer (POSIX). set time | <i>timer_settime()</i> 2-914 |
| | connect user routine to | timer signal. | <i>timer_connect()</i> 2-911 |
| | for timing base/ allocate | timer using specified clock | <i>timer_create()</i> 2-912 |
| | of clock ticks elapsed since | timer was set. return number | <i>envoy_now()</i> 2-170 |
| | functions to be called after | timing. specify | <i>timexPost()</i> 2-919 |
| | to be called prior to | timing. specify functions | <i>timexPre()</i> 2-919 |
| | using specified clock for | timing base (POSIX). /timer | <i>timer_create()</i> 2-912 |
| | break down string into | tokens (ANSI). | <i>strtok()</i> 2-828 |
| | break down string into | tokens (reentrant) (POSIX). | <i>strtok_r()</i> 2-829 |
| | display stack | trace of task. | <i>tt()</i> 2-925 |
| | change | trap-to-monitor character. | <i>tyMonitorTrapSet()</i> 2-931 |
| | whether underlying driver is | tty device. return | <i>isatty()</i> 2-305 |
| | do task-level read for | tty device. | <i>tyRead()</i> 2-932 |
| | do task-level write for | tty device. | <i>tyWrite()</i> 2-933 |
| | initialize | tty device descriptor. | <i>tyDevInit()</i> 2-929 |
| | Motorola MC68302 bimodal | tty driver. | <i>m68302Sio</i> 1-210 |
| | Motorola MC68332 | tty driver. | <i>m68332Sio</i> 1-211 |
| | MC68901 MFP | tty driver. | <i>m68901Sio</i> 1-215 |
| | MB86940 UART | tty driver. | <i>mb86940Sio</i> 1-221 |
| | NS 16550 UART | tty driver. | <i>ns16550Sio</i> 1-273 |
| | Semiconductor SA-1100 UART | tty driver. Digital | <i>sa1100Sio</i> 1-316 |
| | ST 16C552 DUART | tty driver. | <i>st16552Sio</i> 1-371 |
| | ARM AMBA UART | tty driver. | <i>ambaSio</i> 1-9 |
| | initialize | tty driver. | <i>ttyDrv()</i> 2-926 |
| | initialize | tty driver for WDB agent. | <i>wdbVioDrv()</i> 2-1060 |
| | virtual | tty driver support library. | <i>tyLib</i> 1-399 |
| | Motorola MC68360 SCC | tty I/O driver for WDB agent. | <i>wdbVioDrv</i> 1-440 |
| | Motorola MPC800 SMC | UART serial driver. | <i>m68360Sio</i> 1-211 |
| | MB86940 | UART serial driver. | <i>ppc860Sio</i> 1-292 |
| | NS 16550 | UART tty driver. | <i>mb86940Sio</i> 1-221 |
| | Digital Semiconductor SA-1100 | UART tty driver. | <i>ns16550Sio</i> 1-273 |
| | ARM AMBA | UART tty driver. | <i>sa1100Sio</i> 1-316 |
| | all resources used to access | UART tty driver. | <i>ambaSio</i> 1-9 |
| | routines. | UDP group. delete | <i>m2UdpDelete()</i> 2-383 |
| | get UDP MIB-II entry from | UDP information display | <i>udpShow</i> 1-403 |
| | list of listeners. get | UDP list of listeners. | <i>m2UdpTblEntryGet()</i> 2-384 |
| | display statistics for | UDP MIB-II entry from UDP | <i>m2UdpTblEntryGet()</i> 2-384 |
| | initialize | UDP protocol. | <i>udpstatShow()</i> 2-933 |
| | | UDP show routines. | <i>udpShowInit()</i> 2-933 |

| | Keyword | Name | Page |
|--------------------------------|--------------------------------------|--------------------------------|--------|
| user message and send it to | UDP socket. create zbuf from | zbufSockBufSendto() | 2-1079 |
| receive message in zbuf from | UDP socket. | zbufSockRecvfrom() | 2-1081 |
| send zbuf message to | UDP socket. | zbufSockSendto() | 2-1083 |
| packet driver for lightweight | UDP/IP. END based | wdbEndPktDrv | 1-432 |
| packet driver for lightweight | UDP/IP. pipe | wdbPipePktDrv | 1-433 |
| communication functions for | ULIP. initialize WDB agent's | wdbUlipPktDevInit() | 2-1058 |
| communication interface for | ULIP driver. WDB | wdbUlipPktDrv | 1-439 |
| Set debug flag in UNIX's | ULIP driver. | ulipDebugSet() | 2-934 |
| /network interface driver to | ULIP for vxSim for Windows NT. | ntEnd | 1-274 |
| network interfaces/ attach | ULIP interface to list of | ulattach() | 2-934 |
| delete | ULIP interface (VxSim). | ulipDelete() | 2-935 |
| initialize | ULIP interface (VxSim). | ulipInit() | 2-935 |
| interface driver. SMC | Ultra Elite END network | ultraEnd | 1-404 |
| interface driver. SMC Elite | Ultra Ethernet network | if_ultra | 1-171 |
| device. publish | ultra interface and initialize | ultraattach() | 2-937 |
| display statistics for | ultra network interface. | ultraShow() | 2-940 |
| dosFs disk on top of | UNIX. initialize | unixDiskInit() | 2-943 |
| parameters. get NFS | UNIX authentication | nfsAuthUnixGet() | 2-522 |
| parameters. modify NFS | UNIX authentication | nfsAuthUnixPrompt() | 2-522 |
| parameters. set NFS | UNIX authentication | nfsAuthUnixSet() | 2-523 |
| parameters. display NFS | UNIX authentication | nfsAuthUnixShow() | 2-523 |
| set ID number of NFS | UNIX authentication/ | nfsIdSet() | 2-530 |
| create | UNIX BSD 4.3 select library. | selectLib | 1-335 |
| install | UNIX disk device. | unixDiskDevCreate() | 2-942 |
| (VxSim). pass-through (to | UNIX disk driver. | unixDrv() | 2-943 |
| | UNIX) file system library | passFsLib | 1-279 |
| | unix serial driver. | unixSio | 1-407 |
| | unload object module. | VXWModule::~VXWModule() | 2-998 |
| specifying file name or/ | unload object module by | unld() | 2-944 |
| specifying group number. | unload object module by | unldByGroup() | 2-945 |
| specifying module ID. | unload object module by | unldByModuleId() | 2-945 |
| specifying name and path. | unload object module by | unldByNameAndPath() | 2-946 |
| | unmount dosFs volume. | dosFsVolUnmount() | 2-148 |
| | unmount NFS device. | nfsUnmount() | 2-532 |
| make calling task | unsafe from deletion. | taskUnsafe() | 2-890 |
| convert lower-case letter to | upper-case equivalent (ANSI). | toupper() | 2-922 |
| test whether character is | upper-case letter (ANSI). | isupper() | 2-309 |
| lower-case equivalent/ convert | upper-case letter to | tolower() | 2-921 |
| socket. create zbuf from | user data and send it to TCP | zbufSockBufSend() | 2-1078 |
| login prompt and validate | user entry. display | loginPrompt() | 2-341 |
| delete | user entry from login table. | loginUserDelete() | 2-343 |
| WDB | user event library. | wdbUserEvtLib | 1-439 |
| include WDB | user event library. | wdbUserEvtLibInit() | 2-1059 |
| tools.. post | user event string to host | wdbUserEvtPost() | 2-1059 |
| library. | user interface subroutine | usrLib | 1-411 |
| network interface driver for | User Level IP (VxSim). | if_ulip | 1-170 |
| display | user login table. | loginUserShow() | 2-343 |
| library. | user login/password subroutine | loginLib | 1-194 |
| UDP socket. create zbuf from | user message and send it to | zbufSockBufSendto() | 2-1079 |
| set remote | user name and password. | iam() | 2-254 |

| | Keyword | Name | Page |
|--|--|----------------------------------|--------|
| | get current user name and password. | <i>remCurIdGet()</i> | 2-620 |
| | set remote user name and password. | <i>remCurIdSet()</i> | 2-620 |
| | login table. verify user name and password in | <i>loginUserVerify()</i> | 2-344 |
| | connect user routine to timer signal. | <i>timer_connect()</i> | 2-911 |
| | add user to login table. | <i>loginUserAdd()</i> | 2-342 |
| | return from routine using va_list object. /normal | <i>va_end()</i> | 2-954 |
| | va_arg() and/ initialize va_list object for use by | <i>va_start()</i> | 2-954 |
| | get value of task variable. | <i>VXWTask::varGet()</i> | 2-1041 |
| | set value of task variable. | <i>VXWTask::varSet()</i> | 2-1042 |
| | set environment variable. | <i>putenv()</i> | 2-604 |
| | get value of task variable. | <i>taskVarGet()</i> | 2-893 |
| | set value of task variable. | <i>taskVarSet()</i> | 2-895 |
| | get environment variable (ANSI). | <i>getenv()</i> | 2-240 |
| | write string formatted with variable argument list to/ | <i>vsprintf()</i> | 2-972 |
| | write string formatted with variable argument list to fd. | <i>vfdprintf()</i> | 2-956 |
| | write string formatted with variable argument list to/ | <i>vprintf()</i> | 2-971 |
| | initialize environment variable facility. | <i>envLibInit()</i> | 2-169 |
| | remove task variable from task. | <i>VXWTask::varDelete()</i> | 2-1041 |
| | remove task variable from task. | <i>taskVarDelete()</i> | 2-892 |
| | environment variable library. | <i>envLib</i> | 1-106 |
| | add task variable to task. | <i>VXWTask::varAdd()</i> | 2-1040 |
| | add task variable to task. | <i>taskVarAdd()</i> | 2-891 |
| | handle all interrupts in one vector. | <i>z8530Int()</i> | 2-1068 |
| | all DUART interrupts in one vector. handle | <i>m68681Int()</i> | 2-393 |
| | set uninitialized vector handler (ARM). | <i>intUninitVecSet()</i> | 2-283 |
| | x86, MIPS). get interrupt vector (MC680x0, SPARC, i960, | <i>intVecGet()</i> | 2-286 |
| | connect C routine to exception vector (PowerPC). | <i>excConnect()</i> | 2-183 |
| | routine to critical exception vector (PowerPC 403). /C | <i>excCrtConnect()</i> | 2-184 |
| | routine to critical interrupt vector (PowerPC 403). /C | <i>excIntCrtConnect()</i> | 2-187 |
| | /to asynchronous exception vector (PowerPC, ARM). | <i>excIntConnect()</i> | 2-186 |
| | get CPU exception vector (PowerPC, ARM). | <i>excVecGet()</i> | 2-188 |
| | set CPU exception vector (PowerPC, ARM). | <i>excVecSet()</i> | 2-189 |
| | i960,/ write-protect exception vector table (MC680x0, SPARC, | <i>intVecTableWriteProtect()</i> | 2-288 |
| | (MC680x0, SPARC, i960,/ get vector (trap) base address | <i>intVecBaseGet()</i> | 2-284 |
| | (MC680x0, SPARC, i960,/ set vector (trap) base address | <i>intVecBaseSet()</i> | 2-285 |
| | i960, x86, MIPS). set CPU vector (trap) (MC680x0, SPARC, | <i>intVecSet()</i> | 2-286 |
| | initialize exception/interrupt vectors. | <i>excVecInit()</i> | 2-188 |
| | modules. verify checksums on all | <i>moduleCheck()</i> | 2-435 |
| | verify existence of task. | <i>taskIdVerify()</i> | 2-872 |
| | in login table. verify user name and password | <i>loginUserVerify()</i> | 2-344 |
| | facilities (OSPF Opt.). OSPF version 2 (RFC 1583) routing | <i>ospfLib</i> | 1-275 |
| | return BSP version and revision number. | <i>sysBspRev()</i> | 2-843 |
| | print VxWorks version information. | <i>version()</i> | 2-955 |
| | synchronous version of/ | <i>snmpMasterHandlerWR()</i> | 2-783 |
| | translate virtual address for cacheLib | <i>cacheTiTms390VirtToPhys()</i> | 2-76 |
| | translate virtual address for drivers. | <i>cacheDrvVirtToPhys()</i> | 2-56 |
| | address (VxVMI/ translate virtual address to physical | <i>vmTranslate()</i> | 2-971 |
| | driver for WDB agent. virtual generic file I/O | <i>wdbTsfsDrv</i> | 1-435 |
| | Opt.). get OSPF virtual interface entry (OSPF | <i>m2OspfVirtIfEntryGet()</i> | 2-377 |
| | (OSPF Opt.). set OSPF virtual interface entry values | <i>m2OspfVirtIfEntrySet()</i> | 2-378 |

| | Keyword | Name | Page |
|--------------------------------|---------------------------------------|---------------------------------|--------|
| virtual space in shared global | virtual mem (VxVMI Opt.). /to | <i>vmGlobalMap()</i> | 2-963 |
| Opt.). get OSPF | virtual neighbor entry (OSPF | <i>m2OspfVirtNbrEntryGet()</i> | 2-378 |
| virtual/ map physical pages to | virtual space in shared global | <i>vmGlobalMap()</i> | 2-963 |
| map physical space into | virtual space (VxVMI Opt.). | <i>vmMap()</i> | 2-966 |
| agent. | virtual tty I/O driver for WDB | <i>wdbVioDrv</i> | 1-440 |
| change state of block of | virtual memory. | <i>vmBaseStateSet()</i> | 2-959 |
| Opt.). create new | virtual memory context (VxVMI | <i>vmContextCreate()</i> | 2-960 |
| Opt.). delete | virtual memory context (VxVMI | <i>vmContextDelete()</i> | 2-960 |
| Opt.). get current | virtual memory context (VxVMI | <i>vmCurrentGet()</i> | 2-961 |
| Opt.). set current | virtual memory context (VxVMI | <i>vmCurrentSet()</i> | 2-962 |
| (VxVMI Opt.). get global | virtual memory information | <i>vmGlobalInfoGet()</i> | 2-963 |
| (VxVMI Opt.). include | virtual memory show facility | <i>vmShowInit()</i> | 2-968 |
| (VxVMI Opt.). | virtual memory show routines | <i>vmShow</i> | 1-416 |
| initialize base | virtual memory support. | <i>vmBaseLibInit()</i> | 2-958 |
| library. base | virtual memory support | <i>vmBaseLib</i> | 1-413 |
| architecture-independent | virtual memory support library/ | <i>vmLib</i> | 1-414 |
| (VxVMI Opt.). initialize | virtual memory support module | <i>vmLibInit()</i> | 2-965 |
| enable or disable | virtual memory (VxVMI Opt.). | <i>vmEnable()</i> | 2-962 |
| get state of page of | virtual memory (VxVMI Opt.). | <i>vmStateGet()</i> | 2-968 |
| change state of block of | virtual memory (VxVMI Opt.). | <i>vmStateSet()</i> | 2-969 |
| modify mode of dosFs | volume. | <i>dosFsModeChange()</i> | 2-145 |
| unmount dosFs | volume. | <i>dosFsVolUnmount()</i> | 2-148 |
| modify mode of raw device | volume. | <i>rawFsModeChange()</i> | 2-611 |
| disable raw device | volume. | <i>rawFsVolUnmount()</i> | 2-613 |
| modify mode of rt11Fs | volume. | <i>rt11FsModeChange()</i> | 2-661 |
| fragmented free space on RT-11 | volume. reclaim | <i>squeeze()</i> | 2-805 |
| disable tape device | volume. | <i>tapeFsVolUnmount()</i> | 2-864 |
| display dosFs | volume configuration data. | <i>dosFsConfigShow()</i> | 2-140 |
| information. show | volume configuration | <i>cdromFsVolConfigShow()</i> | 2-82 |
| structure. initialize dosFs | volume configuration | <i>dosFsConfigInit()</i> | 2-139 |
| obtain dosFs | volume configuration values. | <i>dosFsConfigGet()</i> | 2-138 |
| block device with raw | volume functions. associate | <i>rawFsDevInit()</i> | 2-610 |
| sequential device with tape | volume functions. associate | <i>tapeFsDevInit()</i> | 2-862 |
| prepare to use raw | volume library. | <i>rawFsInit()</i> | 2-611 |
| initialize tape | volume library. | <i>tapeFsInit()</i> | 2-863 |
| get current dosFs | volume options. | <i>dosFsVolOptionsGet()</i> | 2-147 |
| set dosFs | volume options. | <i>dosFsVolOptionsSet()</i> | 2-147 |
| dosFsDevInit(), specify | volume options for | <i>dosFsDevInitOptionsSet()</i> | 2-142 |
|). specify | volume options for dosFsMkfs(..... | <i>dosFsMkfsOptionsSet()</i> | 2-145 |
| name of shared memory object | (VxMP Opt.). get | <i>VXWSmName::nameGet()</i> | 2-1019 |
| type of shared memory object | (VxMP Opt.). get name and | <i>VXWSmName::nameGet()</i> | 2-1019 |
| in shared-memory name database | (VxMP Opt.). /name string | <i>VXWSmName::nameSet()</i> | 2-1020 |
| shared memory name database | (VxMP Opt.). /object from | <i>VXWSmName::~VXWSmName()</i> | 2-1021 |
| create shared memory partition | (VxMP Opt.). | <i>memPartSmCreate()</i> | 2-427 |
| shared memory message queue | (VxMP Opt.). /and initialize | <i>msgQSmCreate()</i> | 2-466 |
| memory message queue library | (VxMP Opt.). shared | <i>msgQSmLib</i> | 1-252 |
| shared memory binary semaphore | (VxMP Opt.). /and initialize | <i>semBSmCreate()</i> | 2-708 |
| memory counting semaphore | (VxMP Opt.). /shared | <i>semCSmCreate()</i> | 2-710 |
| memory semaphore library | (VxMP Opt.). shared | <i>semSmLib</i> | 1-347 |
| shared memory system partition | (VxMP Opt.). add memory to | <i>smMemAddToPool()</i> | 2-759 |

| | Keyword | Name | Page |
|---------------------------------|--------------------------------------|--------------------------------|-------|
| shared memory system partition | (VxMP Opt.). /for array from | <i>smMemCalloc()</i> | 2-760 |
| shared memory system partition | (VxMP Opt.). /free block in | <i>smMemFindMax()</i> | 2-760 |
| partition block of memory | (VxMP Opt.). /memory system | <i>smMemFree()</i> | 2-761 |
| memory management library | (VxMP Opt.). shared | smMemLib | 1-355 |
| shared memory system partition | (VxMP Opt.). /of memory from | <i>smMemMalloc()</i> | 2-762 |
| shared memory system partition | (VxMP Opt.). /options for | <i>smMemOptionsSet()</i> | 2-762 |
| shared mem system partition | (VxMP Opt.). /of memory from | <i>smMemRealloc()</i> | 2-763 |
| management show routines | (VxMP Opt.). shared memory | smMemShow | 1-357 |
| blocks and statistics | (VxMP Opt.). /system partition | <i>smMemShow()</i> | 2-764 |
| to shared memory name database | (VxMP Opt.). add name | <i>smNameAdd()</i> | 2-765 |
| shared memory object by name | (VxMP Opt.). look up | <i>smNameFind()</i> | 2-766 |
| shared memory object by value | (VxMP Opt.). look up | <i>smNameFindByValue()</i> | 2-767 |
| objects name database library | (VxMP Opt.). shared memory | smNameLib | 1-357 |
| memory objects name database | (VxMP Opt.). /from shared | <i>smNameRemove()</i> | 2-768 |
| name database show routines | (VxMP Opt.). /memory objects | smNameShow | 1-360 |
| memory objects name database | (VxMP Opt.). /of shared | <i>smNameShow()</i> | 2-768 |
| shared memory objects facility | (VxMP Opt.). /calling CPU to | <i>smObjAttach()</i> | 2-772 |
| address to local address | (VxMP Opt.). convert global | <i>smObjGlobalToLocal()</i> | 2-773 |
| memory objects descriptor | (VxMP Opt.). /shared | <i>smObjInit()</i> | 2-774 |
| shared memory objects library | (VxMP Opt.). | smObjLib | 1-361 |
| shared memory objects facility | (VxMP Opt.). install | <i>smObjLibInit()</i> | 2-775 |
| address to global address | (VxMP Opt.). convert local | <i>smObjLocalToGlobal()</i> | 2-776 |
| shared memory objects facility | (VxMP Opt.). initialize | <i>smObjSetup()</i> | 2-776 |
| memory objects show routines | (VxMP Opt.). shared | smObjShow | 1-364 |
| of shared memory objects | (VxMP Opt.). /current status | <i>smObjShow()</i> | 2-777 |
| attempts to take spin-lock | (VxMP Opt.). /of failed | <i>smObjTimeoutLogEnable()</i> | 2-778 |
| driver for User Level IP | (VxSim). network interface | if_ulip | 1-170 |
| (to UNIX) file system library | (VxSim). pass-through | passFsLib | 1-279 |
| to list of network interfaces | (VxSim). /ULIP interface | <i>ulattach()</i> | 2-934 |
| delete ULIP interface | (VxSim). | <i>ulipDelete()</i> | 2-935 |
| initialize ULIP interface | (VxSim). | <i>ulipInit()</i> | 2-935 |
| for/ UNIX-file disk driver | (VxSim for Solaris and VxSim | unixDrv | 1-405 |
| new virtual memory context | (VxVMI Opt.). create | <i>vmContextCreate()</i> | 2-960 |
| delete virtual memory context | (VxVMI Opt.). | <i>vmContextDelete()</i> | 2-960 |
| translation table for context | (VxVMI Opt.). display | <i>vmContextShow()</i> | 2-961 |
| current virtual memory context | (VxVMI Opt.). get | <i>vmCurrentGet()</i> | 2-961 |
| current virtual memory context | (VxVMI Opt.). set | <i>vmCurrentSet()</i> | 2-962 |
| or disable virtual memory | (VxVMI Opt.). enable | <i>vmEnable()</i> | 2-962 |
| virtual memory information | (VxVMI Opt.). get global | <i>vmGlobalInfoGet()</i> | 2-963 |
| in shared global virtual mem | (VxVMI Opt.). /virtual space | <i>vmGlobalMap()</i> | 2-963 |
| initialize global mapping | (VxVMI Opt.). | <i>vmGlobalMapInit()</i> | 2-964 |
| /virtual memory support library | (VxVMI Opt.). | vmLib | 1-414 |
| virtual memory support module | (VxVMI Opt.). initialize | <i>vmLibInit()</i> | 2-965 |
| space into virtual space | (VxVMI Opt.). map physical | <i>vmMap()</i> | 2-966 |
| /page block size | (VxVMI Opt.). | <i>vmPageBlockSizeGet()</i> | 2-967 |
| return page size | (VxVMI Opt.). | <i>vmPageSizeGet()</i> | 2-967 |
| virtual memory show routines | (VxVMI Opt.). | vmShow | 1-416 |
| virtual memory show facility | (VxVMI Opt.). include | <i>vmShowInit()</i> | 2-968 |
| of page of virtual memory | (VxVMI Opt.). get state | <i>vmStateGet()</i> | 2-968 |
| of block of virtual memory | (VxVMI Opt.). change state | <i>vmStateSet()</i> | 2-969 |

| | Keyword | Name | Page |
|--------------------------------------|---|----------------------------|--------|
| write-protect text segment | (VxVMI Opt.) | <i>vmTextProtect()</i> | 2-970 |
| address to physical address | (VxVMI Opt.) /virtual | <i>vmTranslate()</i> | 2-971 |
| wake-up list. | wake up all tasks in <i>select()</i> | <i>selWakeupAll()</i> | 2-706 |
| | wake up task pending in <i>select()</i> | <i>selWakeup()</i> | 2-705 |
| add wake-up node to <i>select()</i> | wake-up list. | <i>selNodeAdd()</i> | 2-704 |
| and delete node from <i>select()</i> | wake-up list. find | <i>selNodeDelete()</i> | 2-705 |
| wake up all tasks in <i>select()</i> | wake-up list. | <i>selWakeupAll()</i> | 2-706 |
| initialize <i>select()</i> | wake-up list. | <i>selWakeupListInit()</i> | 2-706 |
| number of nodes in <i>select()</i> | wake-up list. get | <i>selWakeupListLen()</i> | 2-707 |
| cancel currently counting | watchdog. | <i>VXWwd::cancel()</i> | 2-1046 |
| currently counting | watchdog. | <i>wdCancel()</i> | 2-1061 |
| show information about | watchdog. | <i>wdShow()</i> | 2-1062 |
| initialize | watchdog show facility. | <i>wdShowInit()</i> | 2-1062 |
| | watchdog show routines. | <i>wdShow</i> | 1-441 |
| start | watchdog timer. | <i>VXWwd::start()</i> | 2-1047 |
| construct | watchdog timer. | <i>VXWwd::VXWwd()</i> | 2-1047 |
| construct | watchdog timer. | <i>VXWwd::VXWwd()</i> | 2-1048 |
| destroy | watchdog timer. | <i>VXWwd::~VXWwd()</i> | 2-1048 |
| create | watchdog timer. | <i>wdCreate()</i> | 2-1061 |
| delete | watchdog timer. | <i>wdDelete()</i> | 2-1061 |
| start | watchdog timer. | <i>wdStart()</i> | 2-1063 |
| | watchdog timer class. | <i>VXWwd</i> | 1-429 |
| | watchdog timer library. | <i>wdLib</i> | 1-440 |
| display values of all readable | WD33C93 chip registers. | <i>wd33c93Show()</i> | 2-1054 |
| on SCSI bus (Western Digital | WD33C93 only) /RST line | <i>sysScsiBusReset()</i> | 2-855 |
| /and partially initialize | WD33C93 SBIC structure. | <i>wd33c93CtrlCreate()</i> | 2-1050 |
| Controller library (SCSI-1). | WD33C93 SCSI-Bus Interface | <i>wd33c93Lib1</i> | 1-431 |
| Controller library (SCSI-2). | WD33C93 SCSI-Bus Interface | <i>wd33c93Lib2</i> | 1-431 |
| Controller (SBIC) library. | WD33C93 SCSI-Bus Interface | <i>wd33c93Lib</i> | 1-430 |
| initialize binary semaphore | (WFC Opt.) create and | <i>VXWBSem::VXWBSem()</i> | 2-978 |
| initialize counting semaphore | (WFC Opt.) create and | <i>VXWCSem::VXWCSem()</i> | 2-980 |
| simple linked list class | (WFC Opt.) | <i>VXWList</i> | 1-417 |
| initialize list | (WFC Opt.) | <i>VXWList::VXWList()</i> | 2-986 |
| list as copy of another | (WFC Opt.) initialize | <i>VXWList::VXWList()</i> | 2-987 |
| add node to end of list | (WFC Opt.) | <i>VXWList::add()</i> | 2-981 |
| concatenate two lists | (WFC Opt.) | <i>VXWList::concat()</i> | 2-981 |
| report number of nodes in list | (WFC Opt.) | <i>VXWList::count()</i> | 2-982 |
| extract sublist from list | (WFC Opt.) | <i>VXWList::extract()</i> | 2-982 |
| find node in list | (WFC Opt.) | <i>VXWList::find()</i> | 2-982 |
| find first node in list | (WFC Opt.) | <i>VXWList::first()</i> | 2-983 |
| return first node from list | (WFC Opt.) delete and | <i>VXWList::get()</i> | 2-983 |
| in list after specified node | (WFC Opt.) insert node | <i>VXWList::insert()</i> | 2-984 |
| find last node in list | (WFC Opt.) | <i>VXWList::last()</i> | 2-984 |
| steps away from specified node | (WFC Opt.) /list node nStep | <i>VXWList::nStep()</i> | 2-985 |
| find next node in list | (WFC Opt.) | <i>VXWList::next()</i> | 2-984 |
| find Nth node in list | (WFC Opt.) | <i>VXWList::nth()</i> | 2-985 |
| find previous node in list | (WFC Opt.) | <i>VXWList::previous()</i> | 2-986 |
| specified node from list | (WFC Opt.) delete | <i>VXWList::remove()</i> | 2-986 |
| free up list | (WFC Opt.) | <i>VXWList::~VXWList()</i> | 2-987 |
| mutual-exclusion semaphore | (WFC Opt.) /and initialize | <i>VXWMSem::VXWMSem()</i> | 2-999 |

| | Keyword | Name | Page |
|--------------------------------|------------------------------|-----------------------------------|--------|
| semaphore without restrictions | (WFC Opt.) /mutual-exclusion | <i>VXWMSem::giveForce()</i> | 2-999 |
| memory partition classes | (WFC Opt.) | <i>VXWMemPart</i> | 1-419 |
| create memory partition | (WFC Opt.) | <i>VXWMemPart::VXWMemPart()</i> | 2-992 |
| add memory to memory partition | (WFC Opt.) | <i>VXWMemPart::addToPool()</i> | 2-988 |
| aligned memory from partition | (WFC Opt.) allocate | <i>VXWMemPart::alignedAlloc()</i> | 2-988 |
| block of memory from partition | (WFC Opt.) allocate | <i>VXWMemPart::alloc()</i> | 2-989 |
| largest available free block | (WFC Opt.) find size of | <i>VXWMemPart::findMax()</i> | 2-989 |
| block of memory in partition | (WFC Opt.) free | <i>VXWMemPart::free()</i> | 2-989 |
| get partition information | (WFC Opt.) | <i>VXWMemPart::info()</i> | 2-990 |
| options for memory partition | (WFC Opt.) set debug | <i>VXWMemPart::options()</i> | 2-990 |
| block of memory in partition | (WFC Opt.) reallocate | <i>VXWMemPart::realloc()</i> | 2-991 |
| blocks and statistics | (WFC Opt.) show partition | <i>VXWMemPart::show()</i> | 2-992 |
| object module class | (WFC Opt.) | <i>VXWModule</i> | 1-420 |
| and initialize object module | (WFC Opt.) create | <i>VXWModule::VXWModule()</i> | 2-998 |
| at specified memory addresses | (WFC Opt.) /object module | <i>VXWModule::VXWModule()</i> | 2-995 |
| load object module into memory | (WFC Opt.) | <i>VXWModule::VXWModule()</i> | 2-997 |
| module object from module ID | (WFC Opt.) build | <i>VXWModule::VXWModule()</i> | 2-995 |
| associated with this module | (WFC Opt.) get flags | <i>VXWModule::flags()</i> | 2-993 |
| about object module | (WFC Opt.) get information | <i>VXWModule::info()</i> | 2-993 |
| name associated with module | (WFC Opt.) get | <i>VXWModule::name()</i> | 2-993 |
| find first segment in module | (WFC Opt.) | <i>VXWModule::segFirst()</i> | 2-994 |
| first segment from module | (WFC Opt.) /and return) | <i>VXWModule::segGet()</i> | 2-994 |
| find next segment in module | (WFC Opt.) | <i>VXWModule::segNext()</i> | 2-994 |
| unload object module | (WFC Opt.) | <i>VXWModule::~VXWModule()</i> | 2-998 |
| message queue classes | (WFC Opt.) | <i>VXWMsgQ</i> | 1-421 |
| and initialize message queue | (WFC Opt.) create | <i>VXWMsgQ::VXWMsgQ()</i> | 2-1008 |
| message-queue object from ID | (WFC Opt.) build | <i>VXWMsgQ::VXWMsgQ()</i> | 2-1008 |
| about message queue | (WFC Opt.) get information | <i>VXWMsgQ::info()</i> | 2-1002 |
| number of messages queued | (WFC Opt.) report | <i>VXWMsgQ::numMsgs()</i> | 2-1004 |
| message from message queue | (WFC Opt.) receive | <i>VXWMsgQ::receive()</i> | 2-1004 |
| send message to message queue | (WFC Opt.) | <i>VXWMsgQ::send()</i> | 2-1005 |
| about message queue | (WFC Opt.) show information | <i>VXWMsgQ::show()</i> | 2-1007 |
| delete message queue | (WFC Opt.) | <i>VXWMsgQ::~VXWMsgQ()</i> | 2-1009 |
| ring buffer class | (WFC Opt.) | <i>VXWRingBuf</i> | 1-422 |
| create empty ring buffer | (WFC Opt.) | <i>VXWRingBuf::VXWRingBuf()</i> | 2-1013 |
| object from existing ID | (WFC Opt.) build ring-buffer | <i>VXWRingBuf::VXWRingBuf()</i> | 2-1014 |
| make ring buffer empty | (WFC Opt.) | <i>VXWRingBuf::flush()</i> | 2-1009 |
| of free bytes in ring buffer | (WFC Opt.) determine number | <i>VXWRingBuf::freeBytes()</i> | 2-1010 |
| characters from ring buffer | (WFC Opt.) get | <i>VXWRingBuf::get()</i> | 2-1010 |
| whether ring buffer is empty | (WFC Opt.) test | <i>VXWRingBuf::isEmpty()</i> | 2-1010 |
| buffer is full (no more room) | (WFC Opt.) test whether ring | <i>VXWRingBuf::isFull()</i> | 2-1011 |
| ring pointer by n bytes | (WFC Opt.) advance | <i>VXWRingBuf::moveAhead()</i> | 2-1011 |
| number of bytes in ring buffer | (WFC Opt.) determine | <i>VXWRingBuf::nBytes()</i> | 2-1012 |
| put bytes into ring buffer | (WFC Opt.) | <i>VXWRingBuf::put()</i> | 2-1012 |
| without moving ring pointers | (WFC Opt.) /in ring buffer | <i>VXWRingBuf::putAhead()</i> | 2-1013 |
| delete ring buffer | (WFC Opt.) | <i>VXWRingBuf::~VXWRingBuf()</i> | 2-1014 |
| semaphore classes | (WFC Opt.) | <i>VXWSem</i> | 1-423 |
| object from semaphore ID | (WFC Opt.) build semaphore | <i>VXWSem::VXWSem()</i> | 2-1018 |
| every task pended on semaphore | (WFC Opt.) unblock | <i>VXWSem::flush()</i> | 2-1014 |
| give semaphore | (WFC Opt.) | <i>VXWSem::give()</i> | 2-1015 |

| | Keyword | Name | Page |
|--------------------------------|------------------------------|--|--------|
| reveal underlying semaphore ID | (WFC Opt.) | <code>VXWSem::id()</code> | 2-1015 |
| that are blocked on semaphore | (WFC Opt.) /list of task IDs | <code>VXWSem::info()</code> | 2-1016 |
| information about semaphore | (WFC Opt.) show | <code>VXWSem::show()</code> | 2-1016 |
| take semaphore | (WFC Opt.) | <code>VXWSem::take()</code> | 2-1017 |
| delete semaphore | (WFC Opt.) | <code>VXWSem::~VXWSem()</code> | 2-1018 |
| to all shared memory classes | (WFC Opt.) /behavior common | <code>VXWSmName</code> | 1-425 |
| memory object (VxMP Opt.) | (WFC Opt.) /name of shared | <code>VXWSmName::nameGet()</code> | 2-1019 |
| memory object (VxMP Opt.) | (WFC Opt.) /type of shared | <code>VXWSmName::nameGet()</code> | 2-1019 |
| name database (VxMP Opt.) | (WFC Opt.) /in shared-memory | <code>VXWSmName::nameSet()</code> | 2-1020 |
| name database (VxMP Opt.) | (WFC Opt.) /shared memory | <code>VXWSmName::~VXWSmName()</code> | 2-1021 |
| symbol table class | (WFC Opt.) | <code>VXWSymTab</code> | 1-426 |
| create symbol table | (WFC Opt.) | <code>VXWSymTab::VXWSymTab()</code> | 2-1025 |
| create symbol-table object | (WFC Opt.) | <code>VXWSymTab::VXWSymTab()</code> | 2-1026 |
| table, including group number | (WFC Opt.) /symbol to symbol | <code>VXWSymTab::add()</code> | 2-1021 |
| each entry in symbol table | (WFC Opt.) /to examine | <code>VXWSymTab::each()</code> | 2-1022 |
| look up symbol by name | (WFC Opt.) | <code>VXWSymTab::findByName()</code> | 2-1023 |
| up symbol by name and type | (WFC Opt.) look | <code>VXWSymTab::findByNameAndType()</code> | 2-1023 |
| look up symbol by value | (WFC Opt.) | <code>VXWSymTab::findByValue()</code> | 2-1024 |
| up symbol by value and type | (WFC Opt.) look | <code>VXWSymTab::findByValueAndType()</code> | 2-1024 |
| symbol from symbol table | (WFC Opt.) remove | <code>VXWSymTab::remove()</code> | 2-1025 |
| delete symbol table | (WFC Opt.) | <code>VXWSymTab::~VXWSymTab()</code> | 2-1026 |
| task class | (WFC Opt.) | <code>VXWTask</code> | 1-427 |
| (MC680x0, MIPS, i386/i486) | (WFC Opt.) /status register | <code>VXWTask::SRSet()</code> | 2-1037 |
| create and spawn task | (WFC Opt.) | <code>VXWTask::VXWTask()</code> | 2-1043 |
| initialize task object | (WFC Opt.) | <code>VXWTask::VXWTask()</code> | 2-1043 |
| task with specified stack | (WFC Opt.) initialize | <code>VXWTask::VXWTask()</code> | 2-1045 |
| activate task | (WFC Opt.) | <code>VXWTask::activate()</code> | 2-1027 |
| task without restriction | (WFC Opt.) delete | <code>VXWTask::deleteForce()</code> | 2-1027 |
| create private environment | (WFC Opt.) | <code>VXWTask::envCreate()</code> | 2-1028 |
| retrieve error status value | (WFC Opt.) | <code>VXWTask::errNo()</code> | 2-1028 |
| set error status value | (WFC Opt.) | <code>VXWTask::errNo()</code> | 2-1028 |
| reveal task ID | (WFC Opt.) | <code>VXWTask::id()</code> | 2-1029 |
| get information about task | (WFC Opt.) | <code>VXWTask::info()</code> | 2-1029 |
| check if task is ready to run | (WFC Opt.) | <code>VXWTask::isReady()</code> | 2-1030 |
| check if task is suspended | (WFC Opt.) | <code>VXWTask::isSuspended()</code> | 2-1030 |
| send signal to task | (WFC Opt.) | <code>VXWTask::kill()</code> | 2-1030 |
| name associated with task ID | (WFC Opt.) get | <code>VXWTask::name()</code> | 2-1031 |
| change task options | (WFC Opt.) | <code>VXWTask::options()</code> | 2-1032 |
| examine task options | (WFC Opt.) | <code>VXWTask::options()</code> | 2-1031 |
| change priority of task | (WFC Opt.) | <code>VXWTask::priority()</code> | 2-1033 |
| examine priority of task | (WFC Opt.) | <code>VXWTask::priority()</code> | 2-1032 |
| get task registers from TCB | (WFC Opt.) | <code>VXWTask::registers()</code> | 2-1034 |
| set task's registers | (WFC Opt.) | <code>VXWTask::registers()</code> | 2-1033 |
| restart task | (WFC Opt.) | <code>VXWTask::restart()</code> | 2-1034 |
| resume task | (WFC Opt.) | <code>VXWTask::resume()</code> | 2-1035 |
| contents of task registers | (WFC Opt.) display | <code>VXWTask::show()</code> | 2-1035 |
| task information from TCBs | (WFC Opt.) display | <code>VXWTask::show()</code> | 2-1036 |
| send queued signal to task | (WFC Opt.) | <code>VXWTask::sigqueue()</code> | 2-1037 |
| get task status as string | (WFC Opt.) | <code>VXWTask::statusString()</code> | 2-1038 |
| suspend task | (WFC Opt.) | <code>VXWTask::suspend()</code> | 2-1039 |

| | Keyword | Name | Page |
|--|--------------------------------|---|--------|
| | get task control block | (WFC Opt.). <i>VXWTask::tcb()</i> | 2-1039 |
| | add task variable to task | (WFC Opt.). <i>VXWTask::varAdd()</i> | 2-1040 |
| | remove task variable from task | (WFC Opt.). <i>VXWTask::varDelete()</i> | 2-1041 |
| | get value of task variable | (WFC Opt.). <i>VXWTask::varGet()</i> | 2-1041 |
| | get list of task variables | (WFC Opt.). <i>VXWTask::varInfo()</i> | 2-1042 |
| | set value of task variable | (WFC Opt.). <i>VXWTask::varSet()</i> | 2-1042 |
| | delete task | (WFC Opt.). <i>VXWTask::~VXWTask()</i> | 2-1046 |
| | watchdog timer class | (WFC Opt.). <i>VXWwd</i> | 1-429 |
| | construct watchdog timer | (WFC Opt.). <i>VXWwd::VXWwd()</i> | 2-1047 |
| | construct watchdog timer | (WFC Opt.). <i>VXWwd::VXWwd()</i> | 2-1048 |
| | currently counting watchdog | (WFC Opt.). <i>VXWwd::cancel()</i> | 2-1046 |
| | start watchdog timer | (WFC Opt.). <i>VXWwd::start()</i> | 2-1047 |
| | destroy watchdog timer | (WFC Opt.). <i>VXWwd::~VXWwd()</i> | 2-1048 |
| | test whether character is | white-space character (ANSI). <i>isspace()</i> | 2-308 |
| | initialize | <i>WIN_CHAN</i> . <i>winDevInit()</i> | 2-1064 |
| | initialize | <i>WIN_CHAN</i> , part 2. <i>winDevInit2()</i> | 2-1065 |
| | (SPARC). return contents of | window invalid mask register <i>wim()</i> | 2-1064 |
| | set or display eventpoints | (WindView). <i>e()</i> | 2-149 |
| | stream. read next | word (32-bit integer) from <i>getw()</i> | 2-244 |
| | stream. write | word (32-bit integer) to <i>putw()</i> | 2-605 |
| | one buffer to another one long | word at a time. copy <i>bcopyLongs()</i> | 2-30 |
| | copy one buffer to another one | word at a time. <i>bcopyWords()</i> | 2-31 |
| | flush processor | write buffers to memory. <i>cachePipeFlush()</i> | 2-68 |
| | | write bytes to file. <i>write()</i> | 2-1066 |
| | output stream (ANSI). | write character to standard <i>putchar()</i> | 2-603 |
| | (ANSI). | write character to stream <i>fputc()</i> | 2-219 |
| | (ANSI). | write character to stream <i>putc()</i> | 2-603 |
| | register (MIPS). | write contents of cause <i>intCRSet()</i> | 2-277 |
| | device. | write data to SCSI tape <i>scsiWrtTape()</i> | 2-702 |
| | sequential device. | write file marks to SCSI <i>scsiWrtFileMarks()</i> | 2-701 |
| | do task-level | write for tty device. <i>tyWrite()</i> | 2-933 |
| | buffer (ANSI). | write formatted string to <i>sprintf()</i> | 2-800 |
| | | write formatted string to fd. <i>fdprintf()</i> | 2-195 |
| | standard error stream. | write formatted string to <i>printf()</i> | 2-590 |
| | standard output stream/ | write formatted string to <i>fprintf()</i> | 2-591 |
| | stream (ANSI). | write formatted string to <i>sprintf()</i> | 2-215 |
| | stream (ANSI). | write formatted string to <i>vfprintf()</i> | 2-956 |
| | (ANSI). | write from specified array <i>fwrite()</i> | 2-238 |
| | initiate asynchronous | write (POSIX). <i>aio_write()</i> | 2-13 |
| | block device. | write sector(s) to SCSI <i>scsiWrtSecs()</i> | 2-702 |
| | variable argument list to/ | write string formatted with <i>vsprintf()</i> | 2-972 |
| | variable argument list to fd. | write string formatted with <i>vfprintf()</i> | 2-956 |
| | variable argument list to/ | write string formatted with <i>vprintf()</i> | 2-971 |
| | output stream (ANSI). | write string to standard <i>puts()</i> | 2-604 |
| | | write string to stream (ANSI). <i>fputs()</i> | 2-219 |
| | | write to non-volatile RAM. <i>sysNoRamSet()</i> | 2-853 |
| | stream. | write word (32-bit integer) to <i>putw()</i> | 2-605 |
| | clear entry from cache (68K, | x86). <i>cacheArchClearEntry()</i> | 2-48 |
| | level (MC680x0, SPARC, i960, | x86, ARM). set interrupt <i>intLevelSet()</i> | 2-279 |
| | level (MC680x0, SPARC, i960, | x86, ARM). /interrupt lock-out <i>intLockLevelGet()</i> | 2-282 |

| | Keyword | Name | Page |
|--|----------------------------------|----------------------------------|--------|
| level (MC680x0, SPARC, i960, x86, ARM). /interrupt lock-out | | <i>intLockLevelSet()</i> | 2-282 |
| table (MC680x0, SPARC, i960, x86, ARM). /exception vector | | <i>intVecTableWriteProtect()</i> | 2-288 |
| routine (MC680x0, SPARC, i960, x86, MIPS). /handler for C | | <i>intHandlerCreate()</i> | 2-279 |
| vector (MC680x0, SPARC, i960, x86, MIPS). get interrupt | | <i>intVecGet()</i> | 2-286 |
| (trap) (MC680x0, SPARC, i960, x86, MIPS). set CPU vector | | <i>intVecSet()</i> | 2-286 |
| address (MC680x0, SPARC, i960, x86, MIPS, ARM). /(trap) base | | <i>intVecBaseGet()</i> | 2-284 |
| address (MC680x0, SPARC, i960, x86, MIPS, ARM). /(trap) base | | <i>intVecBaseSet()</i> | 2-285 |
| return contents of | y register (SPARC). | <i>y()</i> | 2-1067 |
| Communications Controller/ Z8530 SCC Serial | | z8530Sio | 1-443 |
| zero out buffer. | | <i>bzero()</i> | 2-46 |
| a time (SPARC). zero out buffer eight bytes at | | <i>bzeroDoubles()</i> | 2-47 |