PERQ
Systems
Corporation

# PERQ OPERATING SYSTEM

## March 1984

This manual is for use with POS Release G.5 and subsequent releases until further notice.


PERQ Systems Corporation
2600 Liberty Avenue
P. O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

TABLE OF CONTENTS

```
module AlignMemory;


AlignMemory - Allocated aligned buffers.
J. P. Strait  29 Sep 81.
Copyright (C) PERQ Systems Corporation.

    Abstract:

        This module allocates buffers which need to be aligned on
        boundaries that are multiples of 256 words.

Version Number V1.2


exports

  type AlignedBuffer = array[0..0] of array[0..255] of Integer;
       AlignedPointer = ^AlignedBuffer;

procedure NewBuffer( var P: AlignedPointer; S, A: Integer );

exception BadAlignment( A: Integer );

procedure NewBuffer( var P: AlignedPointer; S, A: Integer

    Abstract:

        This procedure allocates buffers which must be aligned on
        boundaries that are multiples of 256 words.  A new segment is
        allocated which is somewhat larger than the desired buffer size.
        The segment is set to be unmovable so that the alignment can be
        guaranteed.

    Parameters:

        P - Set to point to a new buffer which is aligned as desired.
        S - Desired size of the buffer in 256 word blocks.
        A - Alignment in 256 word blocks.  That is, 1 means aligned on a
             256 word boundary, 2 means a 512 word boundary, and so on.

    Errors:

        BadAlignment  if A is less than one or greater than 256. BadSize
        (memory manager)  if S is less than one or S+A-1 is greater than
        256. Other memory manager exceptions raised by CreateSegment.
```

```
module AllocDisk;

Written by CMU-people
Copyright (C) 1980 PERQ Systems Corporation
```

Abstract:

Allocdisk allocates and deallocates disk pages. The partition has some number of contiguous pages on it. The number of pages in a partition is specified when the partition is created (using the Partition program). Segments can be created within a partition, e.g. segments may not span partitions. The entire disk can be thought of as a partition (the Root Partition)

A DiskInformationBlock (DiskInfoBlock or DIB) contains all the fixed information about a disk, including its partition names, locations and sizes. It also contains a table used to locate boot segments A disk can be 'mounted' which means that its root partition is known to the system as an entry in the DiskTable.

A Partition Information Block (PartInfoBlock or PIB) contains all of the fixed information about a partition, A partition can also be 'mounted', and this is usually done as part of mounting the disk itself. Partitions mounted are entries in the PartTable. Within a partition, segments are allocated as doubly linked lists of pages

The Free List of a segment is a doubly linked list of free pages. This module maintains this list, as well as the DeviceTable and PartTable It contains procedures for mounting and dismounting disks and partitions, as well as allocating and deallocating space within a partition.

When allocating pages, the module updates the PartInfoBlock every MaxAllocs calls on AllocDisk. Since the system may crash some time between updates, the pointers and free list size may not be accurate.

When a partition is mounted, the pointers are checked to see if they point to free pages. If not, the head of the pointer is found by looking at the "filler" word of the block the free head does point to (which presumably was allocated after the last update of PartInfoBlock). The filler word has a short pointer to the next "free" page, and forms a linked list to the real free list header. Likewise, if the Free tail does not have a next pointer of 0, a deallocate is presumed to have been done since a PartInfoBlock update, and NextAdr pointers are chased to find the real end of the Free List.

```
Version Number V2.9
{**********} exports {**********}

imports Arith from Arith;
imports ReadDisk from ReadDisk;

const
```

```
  MAXDISKS      =  2;  {Floppy and HardDisk}
  MAXPARTITIONS = 10;  {Maximum number of mounted partitions}
  MAXPARTCHARS  =  8;  {how many characters in a partition name}

type

  PartString   = string[MAXPARTCHARS];

  DeviceRecord = record                  {entry in the DeviceTable}
                   InfoBlk: DiskAddr;    {where the DiskInfoBlock is}
                   InUse  : boolean;     {this DeviceTable entry is valid}
                   RootPartition: PartString {name of this disk}
                 end;

  PartRecord   = record                        {entry in the PartTable}
                   PartHeadFree : DiskAddr; {pointer to Head of Free List}
                   PartTailFree : DiskAddr; {pointer to tail of Free List}
                   PartInfoBlk  : DiskAddr; {pointer to PartInfoBlock}
                   PartRootDir  : DiskAddr; {pointer to Root Directory}
                   PartNumOps   : integer;  {how many operations done since
                                             last update of PartInfoBlock}
                   PartNumFree  : FSBit32;  {HINT of how many free pages}
                   PartInUse    : boolean;  {this entry in PartTable is
                                             valid}
                   PartMounted  : boolean;  {this partition is mounted}
                   PartDevice   : integer;  {which disk this partition is
                                             in}
                   PartStart    : DiskAddr; {Disk Address of 1st page}
                   PartEnd      : DiskAddr; {Disk Address of last page}
                   PartKind     : PartitionType; {Root or Leaf}
                   PartName     : PartString {name of this partition}
                 end;
var
  DiskTable   : array [0..MAXDISKS-1]  of DeviceRecord;
  PartTable   : array [1..MAXPARTITIONS] of PartRecord;

procedure InitAlloc;  {initialize the AllocDisk module, called during boot}
procedure DeviceMount(disk: integer);  {mount a disk}
procedure DeviceDismount(disk : integer);  {dismount a disk}
function  MountPartition(name : string) : integer;  {mount a partion,
                                                     return PartTable index}
procedure DismountPartition(name : string);  {dismount a partition}
function  FindPartition(name : string) : integer;  {given a partion name,
                                      look for it in PartTable, return
                                      index}
function  AllocDisk(partition : integer) : DiskAddr;  {allocate a free page
                                                       from a partition}
procedure DeallocDisk(addr : DiskAddr);  {return a page to the free list}
procedure DeallocChain(firstaddr,lastaddr : DiskAddr; numblks : integer);
                                {return a bunch of pages to free list}
function  WhichPartition(addr : DiskAddr) : integer;  {given a Disk Address,
                                      figure out which partition it is in}
procedure DisplayPartitions;  {print the PartTable}

Exception NoFreePartitions;
```

Abstract:

Raised when too many partitions are accessed at one time. The limit is MAXPARTITIONS.

Exception BadPart(msg, partName: String);

Abstract:

Raised when there is something wrong with a partition. This means that the Scavenger should be run.

Parameters:

msg is the problem and partName is the partition name. Print error message as: WriteLn('** ',msg,' for ',partName);

Exception PartFull(partName: String);

Abstract:

Raised when there are no free blocks in a partition to be allocated. This means that some files should be deleted and then the Scavenger should be run.

Parameters:

partName is the full partition

Procedure InitAlloc; Abstract Initialize the AllocDisk module

Side Effects: Sets Initialized to a magic number; sets all InUse and PartInUse to false

Procedure DeviceMount(disk : integer);

Abstract:

Mount the device specified by disk if not already mounted

Parameters:

Disk is a device; it should be zero for HardDisk and 1 for Floppy

Environment: Expects DiskTable to be initialized

Side Effects: Sets the DiskTable for device; loads PartTable with Part names on dev

Errors: Error if no free partition slots in PartTable;

NOTE: No mention is made if device has partitions with names same as those already loaded

```
Procedure DisplayPartitions;

    Abstract:

        Displays information about the current partitions

    Environment: Assumes PartTable and DiskTable set up;

    Calls: AddrToField; IntDouble, WriteLn;

Procedure DeviceDismount(disk : integer);

    Abstract:

        Removes device disk (0 or 1) from DiskTable and removes all its
        partitions

    Parameters:

        Disk is a device (0= HardDisk; 1=Floppy)

    Side Effects: Sets DiskTable[disk].InUse to false and removes all of
        disk's partitions

    Calls: DismountPartition

Function FindPartition(name : string ) : integer;

    Abstract:

        Searches through partition table looking for a partition named
        name; if found; returns its index in the table;

    Parameters:

        name is partition name of form "dev:part>" or ":part>" or "part>"
        where the final ">" is optional in all forms. If dev isn't
        specified then searches through all partition names. If dev is
        specified; then only checks those partitions on that device; name
        may be in any case

    Returns: index in PartTable of FIRST partition with name name (there
        may be more than one partition with the same name in which case it
        uses the oldest one) or zero if not found or name malformed;

    Calls: UpperEqual

    Design: No device name specified is signaled by disk=MAXDISKS;
        otherwise disk is set to be the device which the device part of
        name specifies

Function MountPartition(name : string) : integer;
```

Abstract:

Searches for partition name in part table and mounts it if not mounted already; tries to read the head and tail of free list to see if valid

Parameters:

name is partition name of form "dev:part>" where "dev" and ">" are optional

Returns: index in part Table of partition for name or zero if not found

Side Effects: if not mounted already, then reads in PartInfoBlk and sets partTable fields; tries to read the head and tail of free list to see if valid

Errors: if no free slots for partition then Raises NoFreePartitions if can't find free list head or tail then Raises BadPart

Calls: FindPartition

Procedure DismountPartition(name : string);

Abstract:

Removes partition name from PartTable

Parameters:

name is partition name of form "dev:part>" where "dev" and ">" are optional

Side Effects: Writes out part information in table if partition InUse and mounted

Calls: UpdatePartInfo, ForgetAll

Function AllocDisk(partition: integer) : DiskAddr;

Abstract:

Allocate a free block from partition

Parameters:

Partition is the partition index to allocate the block from

Returns: Disk Address of newly freed block;

Side Effects: Updates the partition info to note block freed; changes header in buffer of block; writes new head of free list with its next and prev fields set to zero and its filler set to next free block; decrements PartNumFree

Errors: Raises PartFull if no free blocks in partition Raises BadPart if free list inconsistent

Calls: ReadHeader, ChangeHeader, FlushDisk, UpdatePartInfo

```
Function WhichPartition(addr : DiskAddr) : integer;
```

Abstract:

Given a disk address; find the partition it is in

Parameters:

addr is a disk address

Returns: index of partition addr falls inside of or zero if none

Calls: DoubleBetween

NOTE: DOESN'T CHECK IF ENTRY IN TABLE IS MOUNTED OR INUSE (*Bug*?)

```
Procedure DeallocDisk(addr : DiskAddr);
```

Abstract:

Returns block addr to whatever partition it belongs to

Parameters:

addr is block to deallocate

Side Effects: adds addr to free list; increments PartNumFree

Calls: AddToTail, WhichPartition, UpdatePartInfo

Procedure DeallocChain(firstaddr,lastaddr : DiskAddr; numblks : integer);

Abstract:

Deallocates a chain of blocks

Parameters:

firstAddr and lastAddr are addresses of blocks to deallocate (inclusive) and numBlks is number of blocks to free

Side Effects: Frees first and last addr using AddToTail; middle blocks not changed

Calls: AddToTail, ChangeHeader, WhichPartition, DoubleAdd, FlushDisk, UpdatePartInfo, DoubleInt

NOTE: No checking is done to see if numBlks is correct

```
module Arith;

Needed until Pascal compiler supports type long.

Copyright (C) 1980 Carnegie-Mellon University

Version Number V2.2


exports

imports FileDefs from FileDefs;  { to get FSBitnn }

type
  MyDouble = packed record
             case integer of
               1:
                 (
                   Lsw : integer;
                   Msw : integer
                 );
               2:
                 (
                   Ptr : FSBit32
                 );
               3:
                 (
                   Byte0 : FSBit8;
                   Byte1 : FSBit8;
                   Byte2 : FSBit8;
                   Byte3 : FSBit8
                 )
             end;

function DoubleAdd(a,b : FSBit32) : FSBit32;
function DoubleSub(a,b : FSBit32) : FSBit32;
function DoubleNeg(a   : FSBit32) : FSBit32;
function DoubleMul(a,b : FSBit32) : FSBit32;
function DoubleDiv(a,b : FSBit32) : FSBit32;
function DoubleInt(a   : integer) : FSBit32;
function IntDouble(a   : FSBit32) : integer;
function DoubleBetween(a,start,stop : FSBit32) : boolean;
function DoubleMod(a,b : FSBit32) : FSBit32;
function DoubleAbs(a   : FSBit32) : FSBit32;
function DblEql(a,b : FSBit32) : boolean;
function DblNeq(a,b : FSBit32) : boolean;
function DblLeq(a,b : FSBit32) : boolean;
function DblLes(a,b : FSBit32) : boolean;
function DblGeq(a,b : FSBit32) : boolean;
function DblGtr(a,b : FSBit32) : boolean;
```

```
Function DoubleAdd( a,b : FSBit32) : FSBit32;
    Abstract:
        Adds two doubles together
    Parameters:
        a and b are doubles to add
    Returns: a+b
Function DoubleSub(a,b : FSBit32) : FSBit32;
    Abstract:
        Subtracts b from a
    Parameters:
        a and b are doubles
    Returns: a-b
    Design: a+(-b)
Function DoubleNeg(a : FSBit32 ) : FSBit32;
    Abstract:
        Does a two-s complement negation of argument
    Parameters:
        a is number to negate
    Returns: -a
Function DoubleAbs(a : FSBit32 ) : FSBit32;
    Abstract:
        Does an absolute value of argument
    Parameters:
        a is number to abs
    Returns: |a|
```

```
Function DoubleMul(a,b : FSBit32) : FSBit32;
    Abstract:
        Multiplies a and b
    Parameters:
        a and b are doubles
    Returns: a*b
Function DoubleDiv(a,b : FSBit32) : FSBit32;
    Abstract:
        Divides a by b
    Parameters:
        a and b are doubles
    Returns: a/b
Function DoubleMod(a,b : FSBit32) : FSBit32;
    Abstract:
        Mods a by b
    Parameters:
        a and b are doubles
    Returns: a mod b
Function DoubleInt(a : integer) : FSBit32;
    Abstract:
        converts a into a double
    Parameters:
        a is integer
    Returns: double of a; if a is negative then does a sign extend
```

```
Function IntDouble(a : FSBit32 ) : integer;
     Abstract:
          returns the low word of a
     Parameters:
          a is a double
     Returns: low word
     Errors: Micro-code raises OvflLI (in Except) if a won't fit in one word
Function DoubleBetween(a,start,stop : FSBit32) : boolean;
     Abstract:
          determines whether a is between start and stop (inclusive)
     Parameters:
          a is double to test; start is low double; stop is high
     Returns: true if a >= start and a <= stop else false
function DblEql(a,b : FSBit32): boolean;
     Abstract:
          determines whether a = b
     Parameters:
          a and b are doubles
     Returns: true if a = b; else false
function DblNeq(a,b : FSBit32): boolean;
     Abstract:
          determines whether a <> b
     Parameters:
          a and b are doubles
     Returns: true if a <> b; else false
```

```
function DblLeq(a,b : FSBit32) : boolean;
     Abstract:
          determines whether a <= b
     Parameters:
          a and b are doubles
     Returns: true if a <= b; else false
function DblLes(a,b : FSBit32) : boolean;
     Abstract:
          determines whether a < b
     Parameters:
          a and b are doubles
     Returns: true if a < b; else false
function DblGeq(a,b : FSBit32) : boolean;
     Abstract:
          determines whether a >= b
     Parameters:
          a and b are doubles
     Returns: true if a >= b; else false
function DblGtr(a,b : FSBit32) : boolean;
     Abstract:
          determines whether a > b
     Parameters:
          a and b are doubles
     Returns: true if a > b; else false
```

```
module BigArea;

Copyright C, 1982, 1983 - PERQ Systems Corporation
```

Abstract:

Provides procedures to allocate and release large segments and groups of segments. For contiguous areas, the mobility can also be set.

```
Version Number V0.5


exports

imports Memory from Memory;

procedure  CreateBigArea (var S: SegmentNumber; TotSize, PieceSize:
                          integer);
procedure  CreateContiguousArea (var S: SegmentNumber;
                                 TotSize: integer; Mob: SegmentMobility);
procedure  DecBigAreaRef (S: SegmentNumber; TotSize, PieceSize: integer);
procedure  DecContiguousAreaRef (S: SegmentNumber; TotSize: integer);
procedure  SortSegList;
function   ConsecutiveSegments(N: integer): SegmentNumber;

exception  BadMobility(M: SegmentMobility);
```

Abstract:

Raised when CreateContiguousArea is given a disallowed Mob value.

```
function ConsecutiveSegments(N: integer): SegmentNumber;
```

Abstract:

Finds a block of N consecutive unallocated segment number, allocates them, and return the segment number of the first.

Parameters:
N - number of segment numbers to allocate.

Returns: The segment number of the first of N previously unallocated segment numbers.

Exceptions:
NoFreeSegments - If no sequence of N free segment numbers can be found.
BadSize - If N<1.

Environment: Interrupts are assumed to be ON initially. They are turned off and then back ON.

Design: First calls SortSegList and then searches free list of segment numbers for a long enough group of segment numbers.

```
procedure  CreateBigArea (var S: SegmentNumber; TotSize,
                PieceSize: integer);
```

Abstract:

Creates a big area, possibly composed of multiple segments. It will be swappable.

Parameters:

S - segment number assigned to area. If the area is more than one piece, consecutive segment numbers are given to the pieces.

TotSize - total number of blocks to allocate to area. (Each block is 256 words.) The maximum value for TotSize is 32767, which corresponds to almost 16 megabytes (more than will fit in all but the largest swapping partition).

PieceSize - Size (in blocks) of each segment to allocate for area. 1 <= PieceSize <= 256. If PieceSize is less than 256, then swapping will be better, but addresses will not be contiguous. Moreover, with a small PieceSize, NoFreeSegments is more likely (because longer sequences of free segment numbers are needed).

Recommended PieceSize: 15 blocks.

Exceptions:

FullMemory - There is not enough room in physical memory to satisfy the request.

NoFreeSegments - Memory manager was unable to find a suitable sequence of consecutive segment numbers.

BadSize - TotSize is negative or 1>PieceSize or 256<PieceSize.

Calls:

ConsecutiveSegments, CreateSegment, MakeEdge, ReleaseSegmentNumber

Environment:

Interrupts are assumed to be ON initially. They are turned off and then back ON.

Design: First finds consecutive segment numbers for the various pieces. Then allocates that many segments and uses those segment numbers. The address of the i'th word is makeptr (S+i div (PieceSize*256), i mod (PieceSize*256), word), but the multiplies and divides can be done with shifting if PieceSize is chosen appropriately. With PieceSize=16 we have the address as makeptr (S+Shift(i,-12), LAnd(i,#7777), word) If i is long and its pieces are i.hi and i.lo then we have makeptr (S+Shift(i.lo,-12)+Shift(i.hi,4), LAnd(i.lo,#7777), word). Note: CreateBigArea should eventually be revised to create segments in the swapped out state. Physical core should only be allocated when the segment is referenced. (Even now, disk space is only assigned when a data segment is to

be swapped out.)

```
procedure  CreateContiguousArea (var S: SegmentNumber;
                TotSize: integer; Mob: SegmentMobility);
```

Abstract:

Creates a contiguous area of physical memory. It is forced to remain resident because of the mobility. Use of this module avoids the swap-out-swap-in of using CreateSegment and SetMobility.

Parameters:

S - segment number assigned to area. If the area is more than one piece, consecutive segment numbers are given to the pieces.
TotSize - total number of blocks to allocate to area. (Each block is 256 words.) Max is 8*256, this is one megabyte.
Mob - Mobility for the segment. Must be UnSwappable or UnMovable.

Exceptions:

FullMemory - There is not enough room in physical memory to satisfy the request.
NoFreeSegments - Memory manager was unable to find a suitable sequence of consecutive segment numbers.
BadMobility - Mob must be UnSwappable or UnMovable.

Environment: Interrupts are assumed to be ON initially. They are turned off and then back ON.

Calls: ContiguousSegments, Compact, FindHole, NewSegmentNumber.

```
procedure  DecBigAreaRef (S: SegmentNumber; TotSize, PieceSize: integer);
```

Abstract:

Releases a big area allocated with CreateBigArea.

Parameters:

S - segment number assigned to area. If the area is more than one piece, consecutive segment numbers are given to the pieces.
TotSize - total number of blocks to allocate to area. (Each block is 256 words.)
PieceSize - Size of each segment to allocate for area. If PieceSize is less than 256, then swapping will be better, but addresses will not be contiguous.

Exceptions:

BadSize - if the size of a segment in the group specified by the above parameters is inconsistent with the size that would have been given by CreateBigArea. others from DecRefCount.

Design: Simply calls DecRefCount for each segment in the group.

procedure DecContiguousAreaRef (S: SegmentNumber; TotSize: integer);

Abstract:

Releases contiguous area allocated by CreateContiguousArea.

Parameters:

S - segment number assigned to area. If the area is more than one piece, consecutive segment numbers are given to the pieces.
TotSize - total number of blocks to allocate to area. (Each block is 256 words.)

Environment: Interrupts are assumed to be ON initially. They are turned off and then back ON.

Calls: DecBigRefArea (with PieceSize=256).

Exceptions: see DecBigAreaRef.

procedure SortSegList;

Abstract:

Sorts the list of segment numbers so it is more likely that consecutive segment numbers can be found.

Design: Sorts the segment number list and reconstructs it, putting longest consective secuence of numbers at the end. Thus shorter consecutive sequecenes are used up first and longer sequences are saved for CreateXxxArea.

Rather than do a sort, the groups of consecutive segment numbers are added to lists categorized by length. The front and rear of each list is in array Group Thus, for 1<=i<=NumGroups-1 all chains of length i are put in the list starting at Group[i].Front and extending along its NextSeg pointers to Group[i].rear; All groups of length >= NumGroups are in the list under Group[NumGroups].

After filing all sequences into Group, a new free list of segment numbers is built with the longest groups at the end.

Environment: Interrupts are assumed to be ON initially. They are turned off and then back ON.

```
module Clock;

Clock - Perq clock routines.
J. P. Strait       1 Feb 81.
Copyright (C) PERQ Systems Corporation, 1981.

     Abstract:

          Clock implements the Perq human-time clock.  Times are represented
          internally by a TimeStamp record which has numeric fields for
          Year, Month, Day, Hour, Minute, and Second.  Times may also be
          expressed by a string of the form YY MMM DD HH:MM:SS where MMM is
          a three (or more) letter month name and HH:MM:SS is time of day on
          a 24 hour clock.

          The clock module exports routines for setting and reading the
          current time as either a TimeStamp or a character string, and
          exports routines for converting between TimeStamps and strings.

Version Number V1.7

exports

imports GetTimeStamp from GetTimeStamp;

const ClockVersion = '1.7';

type TimeString = String;

procedure SetTStamp( Stamp: TimeStamp );
procedure SetTString( String: TimeString);
procedure GetTString( var String: TimeString );
procedure StampToString( Stamp: TimeStamp; var String: TimeString );
procedure StringToStamp( String: TimeString; var Stamp: TimeStamp);

Exception BadTime;

     Abstract:

          Raised when a string passed does not represent a valid time

          procedure GetPERQ2GMT(var Stamp: TimeStamp);
          procedure GetPERQ2Local(var Stamp: TimeStamp);
          procedure PutPERQ2Offset;

exception GTSNotPERQ2;

     Abstract:

          This exception is raised if any of the *PERQ2* procedures are
          called and the current machine is not a PERQ-2.
```

```
exception GTSNoZ80;
```

Abstract:

This exception is raised if GetPERQ2GMT is called and the Z80 does not respond;

```
const OffsetFile = '>HoldOffset.TimeStamp';
```

```
procedure SetTStamp( Stamp: TimeStamp );
```

Abstract:

Sets time to be time specified by Stamp

Parameters:

stamp is new time

SideEffects: Changes current time

```
procedure SetTString( String: TimeString);
```

Abstract:

Sets time to be time specified by String

Parameters:

string is the string of the new time

SideEffects: Changes current time

Errors: Raises BadTime is string is invalid (malformed or illegal time)

```
procedure GetTString( var String: TimeString );
```

Abstract:

Returns the current time as a string

Parameters:

string is the string to be set with the current time

```
procedure StampToString( Stamp: TimeStamp; var String: TimeString );
```

Abstract:

Returns a string for the time specified by stamp

Parameters:

stamp is time to get string for; string is set with time represented by stamp

```
procedure StringToStamp( String: TimeString; var Stamp: TimeStamp );
```

Abstract:

Converts string into a time stamp

Parameters:

string is the string containing time; stamp is stamp set with time according to string

Errors: Raises BadTime is string is invalid (malformed or illegal time)

```
procedure GetPERQ2GMT(var Stamp: TimeStamp);
```

Abstract:

This procedure is used to get the GMT time from the clock chip on the PERQ-2 I/O board.

Parameters:

Stamp is a TimeStamp that will be set to contain the current GMT from the PERQ-2 clock.

Exceptions: if the current IO board is not an EIO then raise GTSNotPERQ2.

```
procedure GetPERQ2Local(var Stamp: TimeStamp);
```

Abstract:

Obtain the local time from the PERQ-2 clock. The local time is generated using the GMT provided by the hardware clock and adding in the time in the offset stored on disk.

Parameters:

Stamp will be set to be the current time stamp. If no local offset was found on the disk then all fileds of Stamp will be -1.

Exceptions: if the current IO board is not an EIO then raise GTSNotPERQ2.

```
procedure PutPERQ2Offset;
```

Abstract:

This procedure is used to write an offset from GMT on the disk. It will read the current System time and create an offset file that gives the offset of the current system time from the GMT returned by hardware clock.

Exceptions: if the current IO board is not an EIO then raise GTSNotPERQ2.

```
module CmdParse;

This module provides a number of routines to help with command parsing.

Written by Don Scelza    April 30, 1980

Copyright (C) 1980 - PERQ Systems Corporation

Version Number V3.7

{********************} Exports {********************}

Const CmdPVersion = '3.5';
      MaxCmds = 30;
      MaxCString = 255;
      CCR = Chr(13);  {same as standard CR}
      CmdChar = Chr(24);
      CmdFileChar = Chr(26);

Type CString = String[MaxCString];
     CmdArray = Array[1..MaxCmds] Of String;

Procedure CnvUpper(Var Str:CString);  {*** USE ConvUpper IN PERQ_String***}
Function UniqueCmdIndex(Cmd:CString; Var CmdTable: CmdArray;
                        NumCmds:Integer): Integer;
Procedure RemDelimiters(Var Src:CString; Delimiters:CString;
                        Var BrkChar:CString);
procedure GetSymbol(Var Src,Symbol:CString; Delimiters:CString;
                    Var BrkChar:CString);

Function NextID(var id: CString; var isSwitch: Boolean): Char;
Function NextIDString(var s, id: CString;var isSwitch: Boolean): Char;

Type pArgRec = ^ArgRec;
     ArgRec = RECORD
                name: CString;
                next: pArgRec;
              END;

     pSwitchRec = ^SwitchRec;
     SwitchRec = RECORD
                   switch: CString;
                   arg: CString;
                   correspondingArg: pArgRec;
                   next: pSwitchRec;
                 END;

Function ParseCmdArgs(var inputs, outputs: pArgRec; var switches:
                      pSwitchRec; var err: String): boolean;
Function ParseStringArgs(s: CString; var inputs, outputs: pArgRec;
                      var switches: pSwitchRec; var err: String): boolean;
Procedure DstryArgRec(var a: pArgRec);
Procedure DstrySwitchRec(var a: pSwitchRec);

Type pCmdList = ^CmdListRec;
     CmdListRec = RECORD
```

```
                    cmdFile: Text;
                    isCharDevice: Boolean;
                    next: pCmdList;
                    seg: Integer;
                  END;

Procedure InitCmdFile(var inF: pCmdList; seg: Integer);
Function DoCmdFile(line: CString; var inF: pCmdList; var err: String):
    boolean;
Procedure ExitCmdFile(var inF: pCmdList);
Procedure ExitAllCmdFiles(var inF: pCmdList);
Procedure DstryCmdFiles(var inF: pCmdList);
Function RemoveQuotes(var s: CString): boolean;

Type ErrorType=
    (ErBadSwitch, ErBadCmd, ErNoSwParam, ErNoCmdParam, ErSwParam,
     ErCmdParam, ErSwNotUnique, ErCmdNotUnique, ErNoOutFile,
     ErOneInput, ErOneOutput, ErFileNotFound, ErDirNotFound,
     ErIllCharAfter, ErCannotCreate, ErAnyError, ErBadQuote);

Procedure StdError(err: ErrorType; param: CString; leaveProg: Boolean);
Function NextString(var s, id: CString; var isSwitch: Boolean): Char;

Procedure InitCmdFile(var inF: pCmdList; seg: Integer);
```

Abstract:

Initializes inF to a valid Text File corresponding to the keyboard. This must be called before any other command file routines. The application should then read from inF^.cmdFile. For example:

```
ReadLn(inFile^.cmdFile, s);
```

or

```
while not eof(inFile^.cmdFile) do ...
```

Use popup only if inF^.next = NIL (means no cmd File). Is a fileSystem file if not inF^.isCharDevice. InF will never be NIL. The user should not modify the pCmdList pointers; use the procedures provided.

Parameters:

InF - is set to the new command list.
seg - the segment number to allocate the command file list out of. If the application doesn't care, use 0. This is useful for programs like the Shell that require the list of command files to exist even after the program terminates. For other applications, use 0.

```
Function DoCmdFile(line: CString; var inF: pCmdList;
          var err: String): boolean;
```

Abstract:

This procedure handles an input line that specifies that a command file should be invoked. The application finds a line that begins with an @ and calls this procedure passing that line. This procedure maintains a stack of command files so that command files can contain other command files. Call InitCmdFile before this procedure.

Parameters:

line - the command line found by the application. It is OK if it starts with an @ but it is also OK if it doesn't.

inF - the list of command files. This was originally created by InitCmdFile and maintained by these procedures. If the name is a valid file, a new entry is put on the front of inF describing it. If there is an error, then inF is not changed. In any case, inF will always be valid.

err - if there is an error, then set to a string describing the error, complete with preceding '** '. If no error, then set to ''. The application can simply do: if not DoCmdFile(s, inF, err) then WriteLn(err);

Returns: True if OK, or false if error.

```
Procedure ExitCmdFile(var inF: pCmdList);
```

Abstract:

Remove top command file from list. Call this whenever come to end of a command file.

Parameters:

InF - the list of command files. It must never be NIL. The top entry is removed from inF unless attempting to remove last command file, when it is simply re-initialized to be the console. It is OK to call this routine even when at the last entry of the list. Suggested use: While EOF(inF^.cmdFile) do ExitCmdFile(inF);

```
Procedure ExitAllCmdFiles(var inF: pCmdList);
```

Abstract:

Remove all command file from list. Use when get an error or a ^SHIFT-C to reset all command files.

Parameters:

InF - the list of command files. It must never be NIL. All entries but the last are removed.

```
Procedure DstryCmdFiles(var inF: pCmdList);
```

Abstract:

Removes all command files from list.

Parameters:

InF - the list of command files. All entries are removed and InF set to NIL.

```
Function NextID(var id: CString; var isSwitch: Boolean): Char;
```

Abstract:

Gets the next word off UsrCmdLine and returns it. It is OK to call this routine when UsrCmdLine is empty (id will be empty and return will be CCR) This procedure also removes comments from s (from to end of line is ignored). This is exactly like NextIDString except it uses the UsrCmdLine by default.

WARNING: Do not mix calls to NextID and RemDelimiters/GetSymbol since the latter 2 may change UsrCmdLine in a way that causes NextID to incorrectly report that an id is not a switch. This procedure also appends a CCR to the end of UsrCmdLine so it should not be printed after this procedure is called.

Parameters:

id - set to the next word on UsrCmdLine. If there are none, then id will be the empty string.

isSwitch - tells whether the word STARTED with a slash "/". The slash is not returned as part of the name.

Results: The character returned is the next "significant" character after the id. The possible choices are "=" "," " " "~" CCR. CCR is used to mean the end of the line was hit before a significant character. If there are spaces after the id and then one of the other break characters defined above, then the break character is returned. If there is a simply another id and no break characters, then SPACE is returned.

SideEffects: Puts a CCR at the end of UsrCmdLine so it is a bad idea to print UsrCmdLine after NextID is called the first time. Removes id and separators from front of UsrCmdLine. The final character is also removed.

Function NextIDString(var s, id: CString; var isSwitch: Boolean): Char;

Abstract:

Gets the next word off s and returns it. It is OK to call this routine when s is empty (id will be empty and return will be CCR) This procedure also removes comments from s (from to end of line is ignored). This is exactly like NextID except it allows the user to specify the string to parse.

WARNING: It is a bad idea to mix calls to NextIDString and RemDelimiters/GetSymbol since the latter 2 may change s in a way that causes NextIDString to incorrectly report that an id is not a switch.

Parameters:

s - String to parse. Changed to remove id and separators from front. The final character is also removed.

id - set to the next word in string. If there are none, then id will be the empty string.

isSwitch - tells whether the word STARTED with a slash "/". The slash is not returned as part of the name.

Results: The character returned is the next "significant" character after the id. The possible choices are "=" "," " " "~" CCR. CCR is used to mean the end of the line was hit before a significant character. If there are spaces after the id and then one of the other break characters defined above, then the break character is returned. If there is a simply another id and no break characters, then SPACE is returned.

Function NextString(var s, id: CString; var isSwitch: Boolean): char;

Abstract:

Gets the next word off s and returns it. It is OK to call this routine when s is empty (id will be empty and return will be CCR) This procedure also removes comments from s (from to end of line is ignored). The character after id is NOT removed from s. This is like NextIDString except the character is removed in NextIDString.

Parameters:

s - String to parse. Changed to remove id and separators from front. Final character is NOT removed.

id - set to the next word in string. If there are none, then id will be the empty string.

isSwitch - tells whether the word STARTED with a slash "/". The slash is not returned as part of the name.

Results: The character returned is the next "significant" character after the id. This character remains at the front of s. The possible choices are "=" "," " " "~" CCR. CCR is used to mean the end of the line was hit before a significant character. If there are spaces after the id and then one of the other break characters defined above, then the break character is returned. If there is a simply another id and no break characters, then SPACE is returned.

Function RemoveQuotes(var s: CString): boolean;

Abstract:

Changes all quoted quotes ('') into single quotes (').

Parameters:

s - string to remove quotes from. It is changed.

Returns: true if all ok. False if a single quote ended the string. In this case s still contains that quote.

Function ParseCmdArgs(var inputs, outputs: pArgRec; var switches: pSwitchRec; var err: String): boolean;

Abstract:

Parses the command line assuming standard form. The command should be removed from the front using NextId before ParseCmdArgs is called.

Parameters:

inputs - set to list describing the inputs. There will always be at least one input, although the name may be empty.

outputs - set to list describing the outputs. There will always be at least one output, although the name may be empty.

switches - set to the list of switches, if any. Each switch points to the input or output it is attached to. This may be NIL if the switch appears before any inputs. If a global switch, the application can ignore this pointer. If a switch is supposed to be local, the application can search for each input and output through the switches looking for the switches that correspond to this arg. Switches may be NIL if there are none.

err - set to a string describing the error if there is one. This string can simply be printed. If no error, then set to ''.

Returns: false if there was a reported error so the cmdLine should be rejected. In this case, the pArgRecs should be Destroyed anyway.

```
Function ParseStringArgs(s: CString; var inputs, outputs: pArgRec;
        var switches: pSwitchRec; var err: String): boolean;
```

Abstract:

Parses the string assuming standard form. The command should be removed from the front using NextId before ParseCmdArgs is called.

Parameters:

s - the string to parse.

inputs - set to list describing the inputs. There will always be at least one input, although the name may be empty.

outputs - set to list describing the outputs. There will always be at least one output, although the name may be empty.

switches - set to the list of switches, if any. Each switch points to the input or output it is attached to. This may be NIL if the switch appears before any inputs. If a global switch, the application can ignore this pointer. If a switch is supposed to be local, the application can search for each input and output through the switches looking for the switches that correspond to this arg. Switches may be NIL if there are none.

err - set to a string describing the error if there is one. This string can simply be printed. If no error, then set to ''.

Returns: false if there was a reported error so the string should be rejected. In this case, the pArgRecs should be Destroyed anyway.

```
Procedure DstryArgRec(var a: pArgRec);
```

Abstract:

Deallocates the storage used by a ArgRec list.

Parameters:

a - the head of the list of ArgRecs to deallocate. It is set to NIL. OK if NIL before call.

```
Procedure DstrySwitchRec(var a: pSwitchRec);
```

Abstract:

Deallocates the storage used by a SwitchRec list.

Parameters:

a - the head of a list of pSwitchRecs to deallocate. It is set to NIL. OK if NIL before call.

```
Procedure StdError(err: ErrorType; param: CString; leaveProg: Boolean);
```

Abstract:

Prints out an error message with a parameter and then optionally exits the user program.

Parameters:

err - the error type found
leaveProg - if true then after reporting error, raises ExitProg to return to the shell. If false then simply returns
param - parameter for the error. The message printed is:

```
ErBadSwitch    - "** <PARAM> is an invalid switch."
ErBadCmd       - "** <PARAM> is an invalid command."
ErNoSwParam    - "** Switch <PARAM> does not take any arguments."
ErNoCmdParam   - "** Command <PARAM> doesn't take any arguments."
ErSwParam      - "** Illegal parameter for switch <PARAM>."
ErCmdParam     - "** Illegal parameter for command <PARAM>."
ErSwNotUnique  - "** Switch <PARAM> is not unique."
ErCmdNotUnique - "** Command <PARAM> is not unique."
ErNoOutFile    - "** <PARAM>  does not have any outputs."
ErOneInput     - "** Only one input allowed for <PARAM>."
ErOneOutput    - "** Only one output allowed for <PARAM>."
ErFileNotFound - "** File <PARAM> not found."
ErDirNotFound  - "** Directory <PARAM> does not exist."
ErIllCharAfter - "** Illegal character after <PARAM>."
ErCannotCreate - "** Cannot create file <PARAM>."
ErBadQuote     - "** Cannot end a line with Quote."
ErAnyError     - "<PARAM>"
```

```
Procedure CnvUpper(Var Str:CString);
```

Abstract:

This procedure is used to convert a string to uppercase.

Parameters:

Str is the string that is to be converted.

Side Effects: This procedure will change Str. ****WARNING**** THIS PROCEDURE WILL SOON BE REMOVED. USED THE PROCEDURE ConvUpper IN PERQ_STRING

```
Function UniqueCmdIndex(Cmd:CString; Var CmdTable: CmdArray;
        NumCmds:Integer): Integer;
```

Abstract:

Does a unique lookup in a command table.

Parameters:
Cmd - the command that we are looking for.
CmdTable - a table of the valid commands. The first valid command in this table must start at index 1.
NumCmds - the number of valid command in the table.

Results: This procedure will return the index of Cmd in CmdTable. If Cmd was not found then return NumCmds + 1. If Cmd was not unique then return NumCmds+2.

```
Procedure RemDelimiters(Var Src:CString; Delimiters:CString;
            Var BrkChar:CString);
```

Abstract:

Removes delimiters from the front of a string.

Parameters:
Src - the string from which we are to remove the delimiters.
Delimiters - a string that contains the characters that are to be considered delimiters.
BrkChar - will hold the character that we broke on.

Side Effects: This procedure will change both Src and BrkChar.

```
Procedure GetSymbol(Var Src,Symbol:CString; Delimiters:CString;
            Var BrkChar:CString);
```

Abstract:

Removes the first symbol from the beginning of a string.

Parameters:
Src - the string from which we are to remove the symbol.

Symbol - a string that is used to return the next symbol.

Delimiters - a string that defines what characters are to be considered delimiters. Any character in this string will be used to terminate the next symbol.

BrkChar - used to return the character that stopped the scan.

Side Effects: Removes the first symbol from Src and places it into Symbol. Places the character that terminated the scan into BrkChar.

```
module Code;

Code.Pas - Common definitions for the Linker and Loader.

J. P. Strait          10 Feb 81.  Rewritten as a module.

Copyright (C) PERQ Systems Corporation, 1981, 1983.

    Abstract:

        Code.Pas defines constants and types shared by the Linker and the
        Loader.  These include definitions of the run file and of offsets
        in the stack segment.

    Design:

        When the format of run files is changed, the constant RFileFormat
        must also be changed.  This is necessary to prevent the procedures
        which read run files from failing.

exports

imports GetTimeStamp from GetTimeStamp;

const   CodeVersion = '1.14';
        RFileFormat = 2;
        QCodeVersion = 4;       { Current QCode Version Number }
        FileLength   = 100;     { max chars in a file name }
        SegLength    = 8;       { max chars in a segment name }
        StackLeader  = 2;       { number of leader words in stack before }
                                { XSTs (must be even)                    }
                                { Currently contains initial TP and GP   }
        DefStackSize = #20;     { default stack segment size (in blocks) }
        DefHeapSize  = #4;      { default heap segment size (in blocks) }
        DefIncStack  = #4;      { default stack size increment (in blocks) }
        DefIncHeap   = #4;      { default heap size increment (in blocks) }
        FudgeStack   = #2000;   { fudge space between system and user GDB's}
                                { this must hold all loader variables at }
                                { maximum configuration in LoadStack }
        CommentLen   = 80;      { the length of comment and version str in
                                  seg}

type
        SNArray = packed array[1..SegLength] of Char;  { segment name }
        pFNString = ^FNString;
        FNString = String[FileLength];  { file name }
        QVerRange = 0..255;             { range of QCode version numbers }

        SegHint = record case Integer of
                    1: (Fid    : Integer;      { file id }
                        Update: TimeStamp);    { update time }
                    2: (Word1 : Integer;
                        Word2 : Integer;
                        Word3 : Integer)
                  end;
```

```
pSegNode = ^SegNode;
pImpNode = ^ImpNode;

{ Segment information record:}

SegNode = record
            SegId    : SNArray;     { segment name }
            RootNam  : pFNString;   { file name without .Pas or .Seg}
            Hint     : SegHint;     { hint to the segment file }
            GDBSize  : integer;     { size of this segment's GDB }
            XSTSize  : integer;     { size of this segment's XST }
            GDBOff   : integer;     { StackBase offset to GDB }
            ISN      : integer;     { segment number inside Linker }
            CodeSize : integer;     { number of blocks in .Seg file }
            SSN      : integer;
            UsageCnt : integer;
            ImpList  : pImpNode;
            Next     : pSegNode
            end;

{ Import information record }

ImpNode = record
            SId      : SNArray;     { name of imported segment }
            FilN     : pFNString;   { file name of imported segment }
            XGP      : integer;     { global pointer of import }
            XSN      : integer;     { internal number of import }
            Seg      : pSegNode;
            Next     : pImpNode
            end;

{ Run file: }

RunElement = (RunHeader,SysSegment,UserSegment,Import,SegFileNames);
RunInfo = record   { run header }
            RFileFormat:integer;
            Version:    integer;
            System:     boolean;
            InitialGP:  integer;
            CurOffset:  integer;
            StackSize:  integer;
            StackIncr:  integer;
            HeapSize:   integer;
            HeapIncr:   integer;
            ProgramSN:  integer;
            SegCount:   integer
            end;

RunFileType = file of Integer;

{ Segment file: }

Language = (Pascal, Fortran, Imp);
pSegBlock = ^SegBlock;
SegBlock = packed record case integer of { .SEG file definition }
             { zeroth (header) block: }
```

```
                0:    (ProgramSegment: boolean;
                       LongIds       : boolean;
                       DbgInfoExists : boolean;
                       OptimizedCode : boolean;
                       SegBlkFiller  : 0..15;
                       QVersion      : QVerRange;
                       ModuleName    : SNArray;
                       FileName      : FNString;
                       NumSeg        : integer;
                       ImportBlock   : integer;
                       GDBSize       : integer;
                       Version       : String[CommentLen];
                       Comment       : String[CommentLen];
                       Source        : Language;
                       PreLinkBlock  : integer;
                       RoutDescBlock : integer;
                       DiagBlock     : integer;
                       QMapFileName  : FNString;
                       SymFileName   : FNString;
                       CompId        : TimeStamp);

                { first block: }
                1:    (OffsetRD     : integer;
                       RoutsThisSeg  : integer);
                2:    (Block: array[0..255] of integer)
                end;

        CImpInfo = record case boolean of   { Import List Info - as      }
                                            { generated by the compiler  }
                       true: ( ModuleName: SNArray;  { module identifier}
                               FileName:   FNString  { file name }
                             );
                       false:( Ary: array [0..0] of integer)
                    end;

        SegFileType = file of SegBlock;
```

```
module ComplexFunctions;

J. B. Brodie     1 Mar 82
Copyright (C) PERQ Systems Corporation 1982.
```

Abstract:

ComplexFunctions implements many of the standard functions whose domain and/or range is within the Complex number system. The implementation of these functions utilize mathematical identies for the relationships between the real number system and the complex number system.

DISCLAIMER: Since Math identities are utilized to evaluate these functions, accuracy and execution speed may be very poor. Only the most cursory testing of these functions has been performed. No guarantees are made as to the accuracy or correctness of the functions. Validation of the functions must be done, but at some later date.

```
Version Number V1.1

exports

type
   Complex = record
      Re : Real;
      Im : Real;
   end;

function CMult   ( Z1,Z2:Complex ) :     Complex;
function CExp    ( Z:Complex ) :         Complex;
function CCos    ( Z:Complex ) :         Complex;
function CSin    ( Z:Complex ) :         Complex;
function CLn     ( Z:Complex ) :         Complex;
function CSqrt   ( Z:Complex ) :         Complex;
function CPowerC ( Z1,Z2:Complex ) :     Complex;
function CPowerR ( Z:Complex; X:Real ) : Complex;

exception CExpReLarge ( Z:Complex );
```

Abstract:

CExpReLarge is raised when CExp is called with a complex number whose real number component (e.g. Z.Re) would cause a result which is too large to be represented on the Perq.

Parameters:

Z -- Argument of CExp

```
exception CExpReSmall ( Z:Complex );
```

Abstract:

CExpReSmall is raised when CExp is called with a complex number whose real number component (e.g. Z.Re) would cause a result which is too small to be represented on the Perq.

Parameters:

Z -- Argument of CExp

```
exception CExpImLarge ( Z:Complex );
```

Abstract:

CExpImLarge is raised when CExp is called with a complex number whose imaginary number component (e.g. Z.Im) would cause a result which is too large to be represented on the Perq.

Parameters:

Z -- Argument of CExp

```
exception CExpImSmall ( Z:Complex );
```

Abstract:

CExpImSmall is raised when CExp is called with a complex number whose imaginary number component (e.g. Z.Im) would cause a result which is too small to be represented on the Perq.

Parameters:

Z -- Argument of CExp

```
exception CCosReLarge ( Z:Complex );
```

Abstract:

CCosReLarge is raised when CCos is called with a complex number whose real number component (e.g. Z.Re) would cause a result which is too large to be represented on the Perq.

Parameters:

Z -- Argument of CCos

```
exception CCosImLarge ( Z:Complex );
```

Abstract:

CCosImLarge is raised when CCos is called with a complex number whose imaginary number component (e.g. Z.Im) would cause a result which is too large to be represented on the Perq.

Parameters:

Z -- Argument of CCos

```
exception CSinReLarge ( Z:Complex );
```

Abstract:

CSinReLarge is raised when CSin is called with a complex number whose real number component (e.g. Z.Re) would cause a result which is too large to be represented on the Perq.

Parameters:

Z -- Argument of CSin

```
exception CSinImLarge ( Z:Complex );
```

Abstract:

CSinImLarge is raised when CSin is called with a complex number whose imaginary number component (e.g. Z.Im) would cause a result which is too large to be represented on the Perq.

Parameters:

Z -- Argument of CSin

```
exception CLnSmall    ( Z:Complex );
```

Abstract:

CLnSmall is raised when CLn is called with a complex number which would cause a result which is too small to be represented on the Perq.

Parameters:

Z -- Argument of CLn

```
exception CPowerZero  ( Z1,Z2:Complex );
```

Abstract:

CPowerZero is raised when either CPowerC or CPowerR is called with a zero exponent.

Parameters:

Z1, Z2 -- Arguments to CPowerC or CPowerR

```
function CMult ( Z1,Z2:Complex ) : Complex;
```

Abstract:

Evaluates the vector cross product of two complex numbers

Parameters:

Z1, Z2 -- Complex numbers to be multiplied

Returns: Complex cross product

```
function CExp ( Z:Complex ) : Complex;
```

Abstract:

Compute exponential function of a complex number

Note: the use of standard type-real functions in order to evaluate this function may artificially constrain this functions Domain.

Domain: Real=[-85.0,87.0]  Imaginary=[-1E5,1E5]
Range:  Real=Imaginary=[RealMLargest,RealPLargest]

Parameters:

Z -- Input argument

Returns: Exponential of Z

```
function CCos ( Z:Complex ) : Complex;
```

Abstract:

Compute the complex Cosine function.

Note: the use of standard type-real functions in order to evaluate this function may artificially constrain this functions Domain.

Domain: Real=[-1E5,1E5]  Imaginary=[-85.0,87.0]
Range:  Real=Imaginary=[RealMLargest,RealPLargest]

Parameters:

Z -- Input argument

Returns: Cosine of Z

```
function CSin ( Z:Complex ) : Complex;
```

Abstract:

Compute the complex Sine function.

Note: the use of standard type-real functions in order to evaluate this function may artificially constrain this functions Domain.

Domain: Real=[-1E5,1E5] Imaginary=[-85.0,87.0] break Range: Real=Imaginary=[RealMLargest,RealPLargest]

Parameters:

Z -- Input argument

Returns: Sine of Z

```
function CLn ( Z:Complex ) : Complex;
```

Abstract:

Compute natural logarithm of a complex number

Note: the use of standard type-real functions in order to evaluate this function may artificially constrain this functions Domain.

Domain: Real=Imaginary=[-1E19,1E19]
Range: Real=[RealMLargest,RealPLargest] Imaginary=[-Pi,Pi]

Parameters:

Z -- Input argument

Returns: Natural log of Z

```
function CSqrt ( Z:Complex ) : Complex;
```

Abstract:

Compute square root of a complex number

Note: the use of standard type-real functions in order to evaluate this function may artificially constrain this functions Domain.

Domain: Real=Imaginary=[-1E19,1E19]
Range: Real=Imaginary=[-1E19,1E19]

Parameters:

Z -- Input argument

Returns: Square root of Z

```
function CPowerC ( Z1,Z2:Complex ) : Complex;
```

Abstract:

Raise an arbitrary complex number to an arbitrary complex power.

Note: the use of standard type-real functions in order to evaluate this function may artificially constrain this functions Domain.

Domain: Real=Imaginary=[-1E19,1E19]
Range: Real=[RealMLargest,RealPLargest] Imaginary=[-Pi,Pi]

Parameters:

Z1 -- Input complex base, Z2 -- Input complex exponent

Returns: Z1 raised to the Z2 power

```
function CPowerR ( Z:Complex; X:Real ) : Complex;
```

Abstract:

Raise an arbitrary complex number to an arbitrary real power.

Note: the use of standard type-real functions in order to evaluate this function may artificially constrain this functions Domain.

Domain: Real=Imaginary=[-1E19,1E19]
Range: Real=[RealMLargest,RealPLargest] Imaginary=[-Pi,Pi]

Parameters:

Z -- Input complex base, X -- Input real exponent

Returns: Z1 raised to the X power

```
module Configuration;

    Abstract:

        Configuration exports a series of functions and variables which
        provide configuration information to POS system and application
        software.

        AUTHOR: C. Beckett
Version Number V0.6

{>>>>>>>>>>>>>>>>>>>>} EXPORTS

Type

  Cf_MonitorType = (Cf_Landscape, Cf_Portrait);
  Cf_IOBoardType = (Cf_CIO, Cf_EIO);

var

  Cf_KeyPad: boolean;
        { If true, the keyboard attached to the Perq has an auxiliary }
        { keypad. }

  Cf_KeyboardStyle: integer;
        { A number which indicates the style of keyboard attached to }
        { the Perq.
        { This number is:
        {     0: If the keyboard is the original one manufactured by PERQ.
        {     1: If the keyboard is the VT100-compatable keyboard
        {        introduced with the Perq K1 ("Kristmas") model in
                 Jan. 1983.}

  Cf_RS232Ports: integer;
        { Holds the number of usable RS232 ports attached to the Perq. }

  Cf_RS232MaxSpeed: integer;
        { Holds an integer which corresponds to the maximum baud rate
        { supported by the the Perq.  This number is coded to conform to
        { the format defined in the system module IO_Unit.            }

  Cf_Monitor: Cf_MonitorType;
        { If Cf_Monitor, the Perq has a landscape monitor with 1024 X 1280
        { resolution.  If Cf_Portrait, the Perq has a monitor with
        { 1024 X 768 resolution. }

  Cf_WCSSize: integer;
        { Code for the number of K words of the writeable control store.
        { 0 = 4Kwcs, 1 = 16Kwcs. }

  Cf_FloatingHardware: boolean;
        { If true, the Perq has an Intel 8087 floating point chip. }

  Cf_BootUnit: integer;
        { A number which indicates the logical unit number of the disk }
```

```
        { drive attached to the Perq from which the system was booted. }
        { 0 = Harddisk, 1 = floppy. }

  Cf_BootChar: char;
        { A letter which indicates the identification of the particular }
        { system (WCS and main memory contents) loaded as part of boot. }

  Cf_IOBoard: Cf_IOBoardType;
        { Indicates the type of IO board attached to the Perq. }
        { This is:
        {
        { Cf_CIO: If the IO board is the 'Current' board for the original
        {         Perq.('CIO board')
        {
        { Cf_EIO: If the IO board is the Ethernet IO board introduced
        {         with the Perq K1 ("Kristmas") model in Jan. 1983.
        {         ('EIO board' }

function Cf_Init: boolean;
                        { Called by IO_Init. Performs initialization
                          logic. }
                        { Should not be called by applications.  }
                        { Returns true unless fatal errors were  }
                        { detected.  }

function Cf_Init: boolean;
```

Abstract:

Called by system before IO_Init is called. Performs initialization logic. Should not be called by applications. Returns true unless fatal errors were detected.

```
module ControlStore;

ControlStore - Load and call routines in the PERQ control-store.

J. P. Strait   ca. July 80.

Copyright (C) PERQ Systems Corporation, 1981, 1983.

     Abstract:

          The ControlStore module exports types defining the format of PERQ
          micro-instructions and procedures to load and call routines in the
          control-store.

Version Number V1.3

exports

type MicroInstruction = { The format of a micro-instruction as produced by
                          the micro-assembler. }
       packed record case integer of
         0: (Word1: integer;
             Word2: integer;
             Word3: integer);
         1: (Jmp: 0..15;
             Cnd: 0..15;
             Z:   0..255;
             SF:  0..15;
             F:   0..3;
             ALU: 0..15;
             H:   0..1;
             W:   0..1;
             B:   0..1;
             A:   0..7;
             Y:   0..255;
             X:   0..255);
         2: (JmpCnd: 0..255;
             Fill1:  0..255;
             SFF:    0..63;
             ALU0:   0..1;
             ALU1:   0..1;
             ALU23:  0..3)
       end;

    MicroBinary = { The format of a micro-instruction and its address as
                    produced by the micro-assembler. }
      record
        Adrs: integer;
        MI:   MicroInstruction
        end;

    TransMicro = { The format of a micro-instruction as needed by the WCS
                   QCode. }
      packed record case integer of
        0: (Word1: integer;
            Word2: integer;
```

```
        Word3: integer);
    1: (ALU23: 0..3;
        ALU0:  0..1;
        W:     0..1;
        ALU1:  0..1;
        A:     0..7;
        Z:     0..255;
        SFF:   0..63;
        H:     0..1;
        B:     0..1;
        JmpCnd:0..255)
    end;

MicroFile = file of MicroBinary;   { A file of micro-instructions. }

procedure LoadControlStore( var F: MicroFile );
procedure LoadMicroInstruction( Adrs: integer; MI: MicroInstruction );
procedure JumpControlStore( Adrs: integer );

exception WCSSizeError;
```

Abstract:

A WCS operation with address greater than 4K was attempted on a system running with only 4K writable control store.

*****WARNING ******

This exception is raised by PASCAL so users using InLineByte do not enjoy this protection.

Resume:

Allowed. The address gets truncated to 12 bits.

```
procedure LoadControlStore( var F: MicroFile );
```

Abstract:

Loads the contents of a MicroFile into the PERQ control-store. The file whould be opened (with Reset) before calling LoadControlStore. It is read to EOF but not closed; thus it should be closed after calling LoadControlStore.

Parameters:

F - The MicroFile that contains the micro-instructions to be loaded.

```
procedure LoadMicroInstruction( Adrs: integer; MI: MicroInstruction );
```

Abstract:

Loads a single micro-instruction into the PERQ control-store.

Parameters:
Adrs - The control store address to be loaded.
MI - The micro-instruction to be loaded.

```
procedure JumpControlStore( Adrs: integer );
```

Abstract:

Transfers control of the PERQ micro engine to a particular address in the control-store.

Note 1: Values may not be loaded onto the expression stack before calling JumpControlStore. If you wish to pass values through the expression stack, the following code whould be used rather than calling LoadControlStore.

```
LoadExpr( LOr( Shift(Adrs,8), Shift(Adrs,-8) ) ); InLineByte( #277
);        the JCS QCode *)
```

Note 2: Microcode called by JumpControlStore should terminate with a "NextInst(0)" microcode jump instruction.

Parameters:
Adrs - The address to jump to.

```
module Convert;

Convert - Conversion functions for reals and longs.
Michael R. Kristofic  25 Feb 82.
Copyright (C) PERQ Systems Corporation, 1981.

    Abstract:

        Functions for converting between floating point and double
        precision integers are provided.

Version Number V1.0

exports

function FloatLong(Arg : Long) : Real;
function TruncLong(Arg : Real) : Long;
function RoundLong(Arg : Real) : Long;

exception R2LOvrFlow(Arg : Real);

    Abstract:


        R2LOvrFlow is raised when RoundLong or TruncLong is called with
        Arg exceeding the range for Longs (-2147483648 .. +2147483647).
        You may resume from this exception in which case RoundLong or
        TruncLong returns -2147483648 or +2147483647.

    Parameters:
        Arg - Argument of RoundLong or TruncLong.


function FloatLong(Arg : Long) : Real;

    Abstract:

        Convert a double precision integer to floating point.

        Domain = [-2147483648, +2147483647].
        Range =   [-2147483648.0, +2147483648.0].

    Parameters:
        Arg - Input value.

    Returns: Floating point equivalent of Arg.

    Design: Zero and -2147483648 are special cased.  Other numbers are
        handled by an algorithm that shifts Arg until it falls into the
        floating point mantissa pattern then sets the exponent based on
        the number and direction of shifts. Floating point negative
        numbers are not 2's complement, so if Arg is negative its 2's
        complement is converted and the result is negated. NOTE: Floating
        point representation can only handle 24 of the possible 31 bits of
        mantissa information, i.e. accuracy is lost for large numbers.
        Rounding occurs in these cases.
```

function TruncLong(Arg : Real) : Long;

Abstract:

Compute the double precision integer equivalent of a floating point number. The fraction part is truncated.

Domain = [-2147483648.0, +2147483520.0].
Range = [-2147483648, +2147483520].

Parameters:
Arg - Input value.

Returns: Double precision integer equivalent of Arg, fraction truncated.

function RoundLong(Arg : Real) : Long;

Abstract:

Compute the double precision integer equivalent of a floating point number. The fraction part is rounded.

Domain = [-2147483648.0, +2147483520.0].
Range = [-2147483648, +2147483520].

Parameters:
Arg - Input value.

Returns: Double precision integer equivalent of Arg, fraction rounded.

```
module DiskDef;

{-----------------------------------------------------------------
{
{ DiskDef -  TV. ( Tony Vezza )
{
{ Copyright (C) 1982, 1983 PERQ Systems Corporation
{
{ Abstract:
{       DiskDef exports variables, constants and Types to the
{       rest of the Pascal Disk SubSystem. Defines the control
{       structures for CIO and EIO disks. Contains a description
{       of the EIO disk uCode to Pascal interface.
{
{-----------------------------------------------------------------}

{$Version V1.6 for POS}

{*****************************}  Exports  {*****************************}

Imports IO_Unit From IO_Unit;
Imports VolumeSystem From VolumeSystem;
Imports DiskIO From DiskIO;


Const
    DIBAddress = 0;                 { Address Of Disk Information Block. }
    WpDB       = 256;               { Words Per Disk Block. }
    WpFS       = 64;                { Words Per Floppy Sector. }
    FSpDB      = WpDB Div WpFS;     { Floppy Sectors Per Disk Block. }
    FSpT       = 24;                { Floppy Sectors Per Track. }
    FFS        = 3;                 { Floppy First Sector. }
    FIdS       = 1;
    FHpS       = WpFS Div 8;        { Word Size ( DiskHeader) }
                                    { Floppy Headers Per Sector. }
    DBpFT      = FSpT Div FSpDB;    { Disk Blocks Per Floppy Track. }

    FirstFC    = 5;                 { First Floppy Cylinder }

    FirstDB    = 30;                { Shuggart - First Disk Block, Length
                                    {  of Boot Rounded to Track Boundary. }

Const
  DskBlockSize = 512;
  DskSPC = 30;          { Shuggart - Sectors per cylinder }
  DskHds = 8;           { Shuggart - Max number of disk heads }
  DskExHds = 0;         { Shuggart - Extra heads not in use }
  DskCyls = 202;        { Shuggart - Number of cylinders }
```

```
Type

{////////////////////////////////////////////////////////////
```

Disk Types and Disk Unit Number

Four Bits are used to designate the Disk Drive type to the uCode and the DiskIO Pascal code. The bits are interpreted in the following manner. Note SMD is not yet defined. Also note the number of spare encodings.

| DiskType Code | Designated Drive Type |
|---|---|
| 0 | Reserved |
| 1 | 5.25 Inch Drive |
| 2 | 14 Inch Drive |
| 3 | 8 Inch Drive |
| 4..15 | Reserved |

Legal Unit Numbers to the uCode are 4 bit quantities. This allows selection of sixteen drives ( Units 0 thru 15).

```
{////////////////////////////////////////////////////////////}

    { Define The HardDisk Types }
    DiskType = ( D5Inch, DUnused, D14Inch, D8Inch,
                 DSMD, DFloppy, DCIOShugart, DCIOMicropolis,
                 Rsvd07, Rsvd06, Rsvd05, Rsvd04,
                 Rsvd03, Rsvd02, Rsvd01, Rsvd00 );


    { Define Unit Number }
    UnitNumber = 0..15;


{////////////////////////////////////////////////////////////
```

Disk Address Formats

There are several Disk Address Formats which are used to access data on the Disks. These are:

VolBlockNumber
PhyVolAddress
LogAddress
OnVolAddress

Each of these is a Long which can be used to Identify a unique block of data on the Disk. Mechanisms (routines) are provided for converting one format address to another format address.

This following table lists relevant Physical Disk Data:

| PlatterSize | DriveType | Capacity | #Cylinders | #Heads | #Sectors |
|---|---|---|---|---|---|
| SMD | CDC 9766 | 300MB | 823 | 19 | 32 |
| SMD | CDC 9767 | 150MB | 411 | 19 | 32 |
| 8 Inch | Micropolis21 | 21MB | 580 | 3 | 24 |
| 8 Inch | Micropolis35 | 35MB | 580 | 5 | 24 |
| 8 Inch | Micropolis70 | 70MB | 1160 | 5 | 24 |
| 14 Inch | SA4000 | 12MB | 202 | 4 | 30 |
| 14 Inch | SA4002 | 24MB | 202 | 8 | 30 |
| 5.25 Inch | Ampex | 20MB | 320 | 8 | 16 |

Here:

| | |
|---|---|
| #Sectors | = Number of Sectors Per Head (or Track) |
| #Heads | = Number of Heads Per Cylinder |
| #Cylinders | = Number of Cylinders Per Drive |

Note that Head(s) is synonomous with Track(s).

VolBlockNumber

Description

In this form a Disk appears to the File System as a linear one dimensional array of Blocks enumerated from 0 to the maximim number of blocks on the Disk. The VolBlockNumber is actually a 19 bit quantity which can be calculated from the PhyVolAddress using the information in the table above:

```
VolBlockNumber = Cylinder * ( #Heads * #Sectors )
                         + Head * #Sectors
                         + Sector
                         - BootSize
```

Here Cylinder, Head and Sector are components of the PhysicalDiskAddress. #Heads and #Sectors are Disk parameters taken from the above table.

Use

All File System transactions to and from a disk use a VolBlockNumber to identify to/from which disk block the transaction will be made.

```
PhyVolAddress
\\\\\\\\\\\\
  Description
    This is a Double with a Word Cylinder Address,
    a Byte Head (or Track) Address and a Byte Sector Address.
    A Physical Volume Address can be calculated from a
    Volume Block Number using the Data in the above table:

    Cyl = (VolBlockNumber+BootSize) Div (#Heads*#Sectors)

    Hd  = ((VolBlockNumber+BootSize) Mod (#Heads*#Sectors))
                        Div  #Sectors

    Sct = (VolBlockNumber+BootSize) Mod #Sectors

    Here again, Cylinder, Head and Sector are components
    of the PhysicalVolAddress. #Heads and #Sectors are
    Disk parameters taken from the above table.
    The operation ( A Div B ) means the Truncated Integer
    Quotient of A divided by B. The operation ( A Mod B )
    means the Remainder of A divided by B.

    For DFloppy:
            PhyDskAdr[0] = Sct
            PhyDskAdr[1] = Cyl

    For DCIOShugart:
            PhyDskAdr[0] = Cyl*(2^8) + Hd*(2^5) + Sct
            PhyDskAdr[1] = 0

    For D8Inch, D5Inch and D14Inch:
            PhyDskAdr[0] = Hd*(2^8) + Sct
            PhyDskAdr[1] = Cyl

  Use
    The Physical Volume Address is the format used to tell
    the uCode which disk address the read, write or seek
    transaction will involve.

LogAddress
\\\\\\\\\
  Description
    The LogAddress is a repackaged form of the
    VolAddress. Packaged in the following way:

    LogAddress = ( VolID * 2^27 ) + ( VolBlockNumber * 2^8 )

OnVolAddress
\\\\\\\\\\\\
  Description
    The OnVolAddress is a repackage form of the
    VolAddress. Packaged in the following way:

    OnVolAddress = 2^31 + 2^30 + ( VolID * 2^27 )
                                  + ( BlockNumber * 2^8 )
```

Use
The OnVolAddress is used as Hints.

{////////////////////////////////////////////////////////////}

```
LogAddress = Long;

PhyVolAddress = Double;
```

{////////////////////////////////////////////////////////////

State Machine Status Bits

Definition of the Status Bits (taken from Hardware Specification):

| | | |
|---|---|---|
| <2:0> | SMSt | Status: |
| 0 | DIdle | State Machine Is Idle. |
| 1 | DBusy | State Machine Is Busy. |
| 2 | DataCRC | CRC Error in Data. |
| 3 | PHMismatch | Physical Header Mismatch. |
| 4 | LHMismatch | Logical Header Mismatch. |
| 5 | HeadCRC | CRC Error in Logical or Physical Header. For Others - Not Used. |
| <3> | SMInt | When Set Bits<2:0> Have Meaning. |
| <4> | NotTrk0orNotSker | For Shugart or 5.25 inch, Used to Find Track 0 For Calibration Of Seeking Algorithm, For Others Indicates an Error While Trying to do a Seek Operation. |
| <5> | NotFault | When Clear This Bit Indicates a Drive Problem. Causes are Specific to each Drive Type. |
| <6> | NotOnCyl | Indicates That a Seek is still in Progress or That the Mechanism has not yet come To Rest. |
| <7> | NotUnitReady | When 0 This Bit indicates that The Selected Drive is Present and Ready to be used. |
| <8> | Index | This Bit Will Toggle (from 1 to 0 or from 0 to 1) For Every Revolution of the Disk. Can Be Useful During Formatting. |

```
        <10:9> DiskTypeCode          Drive Type Identification
            1                            Undefined
            0                            5.25 Inch Drive
            2                            14   Inch Drive
            3                            8    Inch Drive

        <15:11> Unused

{ ////////////////////////////////////////////////////////////}

    SMStatus = Packed Record
          SMSt                : ( DIdle,          { 000007 Bits<2:0> }
                                  DBusy,
                                  DataCRC,
                                  PHMismatch,
                                  LHMismatch,
                                  HeadCRC,
                                  AbnormalError,
                                  SMError );
          SMInt               : Boolean;          { 000010 Bit<3> }
          NotTrkOorNotSker    : Boolean;          { 000020 Bit<4> }
          NotFault            : Boolean;          { 000040 Bit<5> }
          NotOnCyl            : Boolean;          { 000100 Bit<6> }
          NotUnitReady        : Boolean;          { 000200 Bit<7> }
          IndexMark           : Boolean;          { 000400 Bit<8> }
          DkType              : ( Dk5Inch,      { 003000 Bits<10:9> }
                                  DkUnused,
                                  Dk14Inch,
                                  Dk8Inch );
          Unused              : 0..31             { 174000 Bits<15:11> }
      End;


{////////////////////////////////////////////////////////////

    Disk Control Block

      This Data Structure is the Primary Mechanism for communication
      between the DiskIO Pascal System and the Disk Perq uCode.

      The Structure is created for a particular Drive when that
      Drive is mounted.

      The Size of a DCB is 24 Bytes, or 12 Words, or 6 Long Words,
      or 3 Quad Words. A DCB will be Quad aligned in memory to
      facilitate access to it by uCode.

      Components of the DCB are:

      LastHead          <16> Bits
      \\\\\\\\
        Description
            Indicates the current Selection of the Heads.
```

Written By
UCode which Executes the Seek Command and the uCode which executes any implicit seeks needed by any of the other Commands.
Read By
UCode which Executes the Seek Command and the uCode which executes any implicit seeks needed by any of the other Commands.

LastCylinder <16> Bits
\\\\\\\\\\\\
Description
Indicates the current position of the Heads.
Written By
UCode which Executes the Seek Command and the uCode which executes any implicit seeks needed by any of the other Commands.
Read By
UCode which Executes the Seek Command and the uCode which executes any implicit seeks needed by any of the other Commands.

DummyDCBStatus <8> Bits
\\\\\\\\\\\\\\
Description
UnUsed Dummy Field.

TypeDrive <4> Bits
\\\\\\\\\
Description
Indicates physical drive type.
Written By
DiskIO Pascal Code at Mount time.
Read By
UCode uses this data in selecting the correct drive and in formatting the disk address.

UnitNumber <4> Bits
\\\\\\\\\\
Description
Selected Drive for this command. 0..15 are legal.
Written By
By microcode only to Mount Boot Disk.
By DiskIO Pascal Code whenever a command is issued to the uCode.
Read By
uCode to select Drive.

Command <8> Bits, Upper <5> Bits Are 0
\\\\\\\
Description
Command to the uCode. (See Command Descriptions above)
Written By
Pascal DiskIO Routine every time a transaction is performed with the Disk uCode.
Read By

UCode uses this field to determine what to do.

SectorCount <8> Bits
\\\\\\\\\\
Description
On WriteChecks this is the number of sectors to write. On Reads this is the number of sectors to read. For WriteFormat this is the number of Sectors to Format, should be the number of Sectors on a whole Track!!
Written By
DiskIO Pascal Code.
Read By
UCode which executes the Read and WriteFormat Commands.

PhysicalAddress <32> Bits
\\\\\\\\\\\\\\\
Description
Disk address used by the uCode for any read write or seek operation.
Written By
DiskIO Pascal Code.
Read By
uCode to perform the Disk Seek and Access.

HeaderBufferPointer <32> Bits
\\\\\\\\\\\\\\\\\\
Description
Pointer to the memory buffer containing the Logical Header for all compare and write Logical Header operations. For reads this points to the buffer for storing the Logical Header of the Sector being read. Used to perform the DMA.
Written By
Pointer is set up by DiskIO Pascal Code.
Read By
uCode which performs the compare, write Logical Header or read Logical Header operation.

DataBufferPointer <32> Bits
\\\\\\\\\\\\\\\\
Description
Pointer to the Memory Buffer for the Disk Data Transfer. Used to perform the DMA.
Written By
Pointer is set up by DiskIO Pascal Code.
Read By
uCode which initiates the Disk Data transfer.

DskStatus <8> Bits
\\\\\\\\
Description
Indicates the status of the DiskControllerStateMachine.
Written By
Disk uCode.
Read By
DiskIO Pascal code after all transactions with uCode.

```
PhysParameters     <48> Bits
\\\\\\\\\\\\\\
  Description
      Indicates the maximum legal Cylinder, Head and Sector
      addresses (plus 1) for this drive.
      BootSize indicates what number of sectors to skip
      over at Cyl=0, Head=0 and Sector=0 before allocating
      Logical Blocks. This space is reserved for the Boot
      code.
  Written By
      Pascal Mount Procedure.
  Read By
      Pascal and uCode, to format and translate addresses.

Volume Name        <72> Bits
\\\\\\\\\\\
  Description
      Name of the Disk Volume as it appears in the DIB,
      of the Disk. Note the DIB is written by the
      Partition Program when the disk is initialized.
  Written By
      The Mount procedure which will read the DIB and enter
      the DIB.VolName into the DCB.
  Read By
      The procedures: VolIDLookUp and VolNameLookUp.
      They are used by the file system to access
      the Disks.

Dummy          <8> Bits
\\\\\
  Description
      UnUsed Dummy Field.

DCBStatus          <8> Bits
\\\\\\\\
  Description
      Status of this entry in DiskControlArray.
  Written By
      DiskIO Initialization code, and Mount and Dismount
      Pascal Procedures.
  Read By
      Mount and Dismount Pascal Procedures.

PrecompCyl         <16> Bits
  Description
      Cylinder inside of which writes must be precompensated
      for greater bit densities (5.25 inch disk).
  Written By
      Disk Mount code when determining disk type.
  Read By
      Disk IO Microcode

{////////////////////////////////////////////////////////////}

  DStatus = Packed Record
```

```
                Ready       : Boolean;      { Currently UnUsed }
                Free        : Boolean;
                Mounted     : Boolean;
                BootDevice  : Boolean;      { Currently UnUsed }
                InProgress  : Boolean;      { Currently UnUsed }
                Rsvd1       : 0..7          { To Fill Out Byte }
            End;


    EIODskCtrlBlock = Packed Record

            LastHead     : Integer;
            LastCylinder : Integer;        { Cylinder of last Seek, Rd, }
                                           {  RdCheck, WrFormat, Wr }
                                           {  or WrCheck operation. }

            DummyDCBStatus       : 0..255;

            DskType      : DiskType;           { 0..15 }
            DskUnit      : UnitNumber;         { 0..15 }

            Command      : 0..255;      { Legal DiskuCodeCommand = 0..7 }

            SectorCount  : 0..255;           { Only used during Reads. }

            PhysicalAddress     : PhyVolAddress;   { Double }

            HeaderBufferPointer : IOHeadPtr;       { Long }
            DataBufferPointer   : IOBufPtr;        { Long }

            DskStatus    : SMStatus;           { Word }

            PhysParameters      : Packed Record       { 5 Words }
                    Sector          : 0..255;
                    Head            : 0..255;
                    Cylinder        : Integer;
                    BootSize        : Integer;
                    DiskPages       : Long
                End;

            VolumeName   : VolName;          { String[8], or 9 Bytes }

            DummyByte    : 0..255;

            DCBStatus : DStatus;

            PrecompCyl : Integer;

        End;  { Case }

{////////////////////////////////////////////////////////////
```

Disk Control Array

The Disk Control Array is an array of DCB's. For every disk which has been Mounted there is an active DCB. When a Disk is

```
    Dismounted the active DCB for that Disk is made inactive.

    When a StartIO is issued to the uCode, it is given (on the
    E Stack) a VolumeID. This VolumeID is actually an Index to
    this array. The uCode then Procedes to determine what command
    to execute by examining the indexed (selected) DCB. All the
    information necessary for executing this command and returning
    results is contained in this DCB. In fact some information is
    also returned in the DCB by the uCode.

{////////////////////////////////////////////////////////////}

  DskCtrlArray = Packed Array [VolRangeType] Of EIODskCtrlBlock ;

  PtrDskCtrlArray = ^DskCtrlArray;

{////////////////////////////////////////////////////////////

 Commands to the EIO uCode

  The uCode will be capable of performing the following
  operations:


  Idle (ReadStatus) Command
  \\\\\\\\\\\\\\\\\\\\\\\\
    Description
        Selects the given Drive, then returns the current
        State Machine Status and puts the State Machine in
        an Idle Loop.
    Use
        Can be used to get the Status of a Drive and
        Status of the State Machine.
    Inputs
        Items of the DiskControlBlock Which are used:
              Command = Idle
              DiskType          :  To Select Drive
              Unit Number       :  To Select Drive
    Outputs
        SMState Returned in DiskControlBlock after Unit
        is selected.

  WrCheck Command
  \\\\\\\\\\\\\\\
    Description
        This Command is used to check a Logical Header
        and write a Data Buffer to a Disk block. An
        implicit Seek is performed to the desired disk
        address. Single and Multi Sector WriteChecks are
        currently supported.
    Use
        This command can be used to write block(s)
        of an existing file without modifying the overall
        blocks utilized by the file. Used by the System
        System for Swapping.
```

Inputs

Items of the DiskControlBlock Which are used:

Command = WriteCheck

| | | |
|---|---|---|
| SectorCount | : | Number of Sectors to Write. |
| DiskType | : | To Select Drive. |
| DiskType | : | To Select Drive. |
| Unit Number | : | To Select Drive. |
| LastCylinder | : | To Perform Seek. |
| PhysicalAddress | : | Disk Address to WriteCheck. |
| HeaderPointer | : | Address of Logical Header. For DMA and Compare. |
| DataBufferPointer | : | Address of Data Buffer. For DMA. Data written onto the Disk. |

Outputs

| | | |
|---|---|---|
| Status | : | SMStatus Reg after WriteCheck is complete. |
| LastCylinder | : | Current Cylinder designated by the PhysicalAddress. |

WrFirst Command

\\\\\\\\\\\\\\\

Description

This Command is used to write a Logical Header and write a Data Buffer to a Disk block. An implicit Seek is performed to the desired disk address. No Checking of the old (on disk) Logical Header is performed. This function is used by the file system as FirstWrite. Only single Sector Writes are currently supported.

Use

This command is used to create new files and modify existing files. This command is used to perform File System Disk Writes.

Inputs

Items of the DiskControlBlock Which are used:

Command = Write

| | | |
|---|---|---|
| DiskType | : | To Select Drive. |
| Unit Number | : | To Select Drive. |
| LastCylinder | : | To Perform Seek. |
| PhysicalAddress | : | Disk Address to Write. |
| HeaderPointer | : | Address of Logical Header. For DMA. Logical Header is Written onto the Disk |
| DataBufferPointer | : | Address of Data Buffer. For DMA. Data is written onto the Disk. |

Outputs

| | | |
|---|---|---|
| Status | : | SMStatus Reg after Write is complete. |
| LastCylinder | : | Current Cylinder designated by the PhysicalAddress. |

Format Command
\\\\\\\\\\\\\\

Description

This command writes Physical Header, Logical Header and Data Block to the selected Disk Sectors. This command is used to format an entire track. The uCode must first seek the correct Cylinder. Select the appropriate track. Then wait for the Index pulse from the drive. When the Index is seen then the uCode must WriteFormat the number of Sectors given by SectorCount. Note that the Sector number of the PhysicalHeader must be initially cleared and incremented after each sector is transferred. The old data in the Disk Track is Lost. No provision is provided for formatting a single Sector in the middle of a track. It is hard to imagine how or why this would be done.

Use

The Command is used only in the Disk Formatting and Disk Initialization Procedures.

Inputs

Items of the DiskControlBlock Which are used:

Command = FormatWrite

| | | |
|---|---|---|
| DiskType | : | To Select Drive. |
| Unit Number | : | To Select Drive. |
| LastCylinder | : | To Perform Seek. |
| PhysicalAddress | : | Disk Address to Format. Initially the Physical Header to Write. Sector Number will be cleared then incremented by uCode. |
| SectorCount | : | Number of Sectors to Format. |
| HeaderPointer | : | Address of Logical Header. For DMA. Logical Header which is written. The same Logical Header is written to all Sectors to be Formatted by one command. |
| DataBufferPointer | : | Address of Data Buffer. For DMA. Data written onto the Disk. The same Data Block is written to all Sectors which are Formatted by one command. |

Outputs

| | | |
|---|---|---|
| Status | : | SMStatus Reg after Write is complete. |
| LastCylinder | : | Current Cylinder designated by the PhysicalAddress. |

RdCheck Command
\\\\\\\\\\\\\\\

Description

This command compares Logical Header Data given as an argument with the Logical Header of the selected sector. The Logical Header and Data Block are read off the Disk. An implicit Seek is performed to the desired Cylinder. Multiple Sector Reads are supported by the uCode. When doing a Multiple Sector Read the first Sector is address by the PhysicalAddress. Subsequent Sectors are addressed by the LogicalHeaders of their Predecessor. The DataBuffer is assumed to be large enough to hold the Multiple Sector transfer. The HeaderBuffer is overwritten with the Logical Header of each Sector which is transferred. Thus it is only the size of one LogicalHeader. When the transfer is complete the HeaderBuffer will be the LogicalHeader of the last Sector Transferred.

Use

Not used to perform reads of files by the File System. Note that the contents of the Disk Sector's Logical Header must be known for this operation to succeed. Used by the System for Swapping.

Inputs

Items of the DiskControlBlock Which are used:

Command = ReadCheck

| | | |
|---|---|---|
| SectorCount | : | Number of Sectors to Read. |
| DiskType | : | To Select Drive. |
| Unit Number | : | To Select Drive. |
| LastCylinder | : | To Perform Seek. |
| PhysicalAddress | : | Disk Address to ReadCheck. |
| HeaderPointer | : | Address of Logical Header. For DMA and Compare. Logical Header of the Sector will go here. |
| DataBufferPointer | : | Address of Data Buffer. For DMA. Data will be Read from the disk into this buffer. |

Outputs

| | | |
|---|---|---|
| HeaderPointer | : | HeaderBuffer will be loaded with the Logical Header of the Last Sector which is Read. |
| DataBufferPointer | : | DataBuffer will be loaded with the Data Block of the Sector(s) being Read. |
| Status | : | SMStatus Reg after ReadCheck is complete. |
| LastCylinder | : | Current Cylinder designated by the PhysicalAddress. |

DiagRead Command
\\\\\\\\\\\\\\\

Description

Read the Logical Header and Data Block of the specified Disk Sector. No checking of the Logical Header is performed. An Implicit Seek is performed to the selected Disk Cylinder. Note that Multiple Sector Reads are supported by the uCode. When doing a Multiple Sector Read the first Sector is address by the PhysicalAddress. Subsequent Sectors are addressed by the LogicalHeaders of their Predecessor. The DataBuffer is assumed to be large enough to hold the Multiple Sector transfer. The HeaderBuffer is overwritten with the Logical Header of each Sector which is transferred. Thus it is only the size of one LogicalHeader. When the transfer is complete the HeaderBuffer will be the LogicalHeader of the last Sector Transferred.

Use

Used when the Logical header of a Disk Sector is not known, but the Sector needs to be read, for example, in Scavenge.

Inputs

Items of the DiskControlBlock Which are used:

Command = Read

| | | |
|---|---|---|
| DiskType | : | To Select Drive. |
| Unit Number | : | To Select Drive. |
| LastCylinder | : | To Perform Seek. |
| PhysicalAddress | : | Disk Address to Read. When Reading more than one Sector this is only the address of the first. |
| SectorCount | : | Number of Sectors to Read. |
| HeaderPointer | : | Address of Logical Header. Logical Header of the Sector will go here. |
| DataBufferPointer | : | Address of Data Buffer. For DMA. Data will be Read from the disk into this buffer. |

Outputs

| | | |
|---|---|---|
| HeaderPointer | : | HeaderBuffer will be loaded with the Logical Header of the last Sector Read. |
| DataBufferPointer | : | DataBuffer will be loaded with the Data Block of all the Sectors Read. |
| Status | : | SMStatus Reg after Read is complete. |
| LastCylinder | : | Cylinder of the last block which is read. |

```
    Seek Command
    \\\\\\\\\\\
      Description
          Move the Heads of the specified Drive to the
          specified Cylinder and Head Addresses.
      Use
          Used in Initialization, Mounting and Dismounting,
          and in Error Recovery. Used to determine size of
          some Disk types available in different sizes.
          Not used by the File System. The commands Write,
          WriteCheck, Read and ReadCheck all perform implicit
          seeks in the Disk uCode.
      Inputs
          Items of the DiskControlBlock Which are used:
              Command = Seek
              DiskType          :  To Select Drive.
              Unit Number       :  To Select Drive.
              LastCylinder      :  To Perform Seek.
              PhysicalAddress   :  Disk Address to Seek. Only
                                     the Cylinder and Head
                                     Addresses are used.
                                     used.
      Outputs
              Status            :  SMStatus Reg after
                                     Read is complete.
              LastCylinder      :  Cylinder to which Seek
                                     has been done.

    Reset Command
    \\\\\\\\\\\\
      Description
          Clears all error conditions in the controller and
          in the selected drive. Also performs an implied
          Seek to Cylinder 0. The uCode will Issue a Drive
          Restore Command.
      Use
          Used at initialization, Mounting, Dismounting and
          Error Recovery.
      Inputs
          Items of the DiskControlBlock Which are used:
              Command = Reset
              DiskType          :  To Select Drive.
              Unit Number       :  To Select Drive.
      Outputs
              Status            :  SMStatus Reg after
                                     Seek to 0 is complete.
              LastCylinder      :  Set to 0.

{////////////////////////////////////////////////////////////}

Type
  DskCmds = (DskIdle, DskRdCheck, DskDiagRead, DskWrCheck,
             DskWrFirst, DskFormat, DskSeek, DskClear);
```

```
  PDiskCtrlBlock = ^DiskCtrlBlock;
  DiskCtrlBlock = Packed Record      { This must be quad word aligned }
      Buffer       : IOBufPtr;       { Pointer to data buffer for
                                       transaction }
      DskCommand   : 0..255;
      DskNumSect   : 0..255;
      DskAddr      : Integer;        {for icl cio microp holds hd/sec}
      DskHeader    : IOHeader;
      Dsk          : DskResult;
      DskpNext     : PDiskCtrlBlock;
      DskDevCyl    : Integer;        {Added for icl cio microp,
                                      dev (3 bits) and cyl (13) }

    End;


Type

    DIBlock = Packed Record

        BootSize : Integer;                  { blocks in boot }
        NumSector : Integer;
        NumHeads : Integer;
        NumCylinders : Integer;
        PreCompCylinder : Integer;
        DIB1Filler : array[1..109] of integer;  { Word }

        { String to use as name for this volume after mount. }
        VolumeName  : packed array[1..8] of char;

        { Hints of the addresses of the first and last logical }
        {  blocks of this volume. }
        VolumeStart: Long;
        VolumeEnd : Long;

        { Hints of the Partition Information Blocks for the }
        {  partitions of this volume. }
        SubParts  : array[0..63] of Long;

        DIB2Filler : array[1..2] of integer; { Word }

        case DeviceClass : DiskKinds of
                      FlpDisk : ( );
                      IntDisk : (IntDiskClass : IntDiskKinds);
                      ExtDisk : ( )

      End; { DIBlock }

    PDIBlock = ^DIBlock;

{////////////////////////////////////////////////////////////

    When Mapping this File System stuff onto the Floppy, FSpDB
    Floppy Sectors are used for each Disk Page. One Sector of each
    Track of the Floppy is used to hold all the Headers for the
```

```
    Pages that fit in the remaining Sectors of that Track. One
    Floppy Sectors per Track are not used.

{////////////////////////////////////////////////////////////}

Type
    FlopHdArray = Array [ 0..FHpS-1] of IOHeader; { Holds the Headers }
    FlopHeadPtr = ^FlopHdArray;


Var
    PtrDCA          : PtrDskCtrlArray;
    NumDCBUsed      : VolRangeType;
    IntDiskType     : IntDiskKinds;
    PtrVBuf         : PtrVolBuffer;
    PtrVHBuf        : PtrVolHeaderBuffer;
    StatPtr         : IOStatPtr;
    BufPtr          : IOBufPtr;   { note that a full disk buffer is not
                                    allocated for this variable,
                                    Use only with IOSeek, IOIdle, IOReset }
    HdrPtr          : IOHeadPtr;
    FHeadPtr        : FlopHeadPtr;


Var
    EIOFlag         : Boolean;
    PDskCtrl : PDiskCtrlBlock;
    Initialized : Integer;
    CIODiskType : (CIOShugart, CIOMicropolis, CIOUnknown);
```

```
module DiskIO;


{-----------------------------------------------------------------
{
{ Abstract
{
{ This is an implementation of the DiskIO interface which uses the
{ new VolumeSystem module.  It is provided as a compatibility module
{ in order to support the k1 hardware with little modification of system
{ software above the level of DiskIO.  Eventually diskio will be removed
{ from the system and higher level software will need to be modified to
{ use VolumeSystem directly; in particular, this will be required to
{ support more than a single hard disk and single floppy.
{
{ Old Abstract:
{
{ This module implements the basic low level operations to disk devices.
{ It services the Hard Disk and the Floppy.  When dealing with the floppy
{ here, the structures on the hard disk are mapped to the structures
{ on the floppy.
{
{-----------------------------------------------------------------}

{$Version 4.8 for POS}

{**********} exports {**************************}

imports FileDefs from FileDefs;
imports IOErrors from IOErrors;

const
    HARDNUMBER          = 0;            {device code of Shugart Disk}
    FLOPPYNUMBER        = 1;            {device code of FloppyDisk}

    {a Disk Address can be distinguished from a Segment Address by the
      upper two bits (in 32 bits).  These bits have a nonzero code to
      which disk the address is part of}

    RECORDIOBITS        = #140000;      {VirtualAddress upper 16 bits of
                                            disk}
    DISKBITS            = RECORDIOBITS + (HARDNUMBER*(#20000));
    FLOPBITS            = RECORDIOBITS + (FLOPPYNUMBER*(#20000));

    {The following definitions tell how many entries there are in the
     three pieces of the random index.  The first piece (Direct)
     are blocks whose DiskAddresses are actually contained in the
     Random Index (which is part of the FileInformationBlock).
     The second section has a list of blocks each of which contain
     128 Disk Addresses of blocks in the file, forming a one level
     indirect addressing scheme.

     For very large files, the third section (DblInd) has DiskAddresses of
     blocks which point to other blocks which contain 128 DiskAddresses
     of blocks in the file, forming a two level indirect scheme.}
```

```
    DIRECTSIZE     =  64; { Entries in FIB of blocks directly accessible }
    INDSIZE        =  32; { Entries in FIB of 1 level indirect blocks }
    DBLINDSIZE     =   2; { Entries in FIB of 2 level indirect blocks }

    FILESPERDIRBLK = 16;  { 256 / SizeOf(DirEntry) }
    NUMTRIES       = 15;  { number of tries at transfer before aborting }

type
    {Temporary segments go away when processes are destroyed,
     Permanent segments persist until explicitly destroyed
     Bad Segments are not well formed segments which are not
     readable by the Segment system}

    SpiceSegKind  = (Temporary, Permanent, Bad);
    PartitionType = (Root, UnUsed, Leaf); {A Root Partition is a device}
    DeviceType    = (Winch12, Winch24, FloppySingle, FloppyDouble,
                     CIOMicrop, Generic5Inch);

    MyDble = Array [0..1] of integer;

    DiskCheatType = record
                      case integer of
                        1: (
                             Addr    : DiskAddr
                           );
                        2: (
                             Dbl     : MyDble { should be IO.Double but
                                                don't import IO in export
                                                section }
                           );
                        3: (
                             Seg     : SegID
                           );
                        4: (
                             Lng     : FSBit32
                           )
                      end;

    { A directory is an ordinary file which contains SegIDs of files
      along with their names.  Directories are hash coded by file name
      to make lookup fast.  They are often sparse files (ie contain
      unallocated blocks between allocated blocks).  The file name is a
      SimpleName, since a directory can only contain entries for
      files within the partition (and thus device) where the directory
      itself is located }

    DirEntry   = packed record
                   InUse    : boolean; {true if this DirEntry is valid}
                   Deleted  : boolean; {true if entry deleted but not
                                        expunged}
                   Archived : boolean; {true if entry is on backup tape}
                   UnUsed   : 0..#17777;   {reserved for later use}
                   ID       : SegID;
                   Filename : SimpleName
                 end;
```

```
    { The fillers in headers which are used as hints to the new head of
      the free list when a block is allocated in a partition (by
      AllocDisk) have been represented as unsigned 16 bit numbers
      denoting disk relative logical block numbers.  For disks larger
      than 64K blocks this is not possible and a partition relative
      denotation is used instead on all future disks.  FillerSemantics
      is the type of a field in the DIB of a disk which tells which
      interpretation applies for that disk. }

    FillerSemantics = (DiskRelative, PartRelative);

    DiskBuffer = packed record
                   case integer of
                     1: (
                          Addr : array [0..(DISKBUFSIZE div 2)-1] of
                            DiskAddr
                        );
                     2: (
                          IntData : array [0..DISKBUFSIZE-1] of FSBit16
                        );
                     3: (
                          ByteData : packed array [0..DISKBUFSIZE*2-1] of
                            FSBit8
                        );
  {4 is format of the FileInformationBlock; the FIB has Logical Block -1 }
                     4: (

                          FSData      : FSDataEntry;

                          {The Random Index is a hint of the DiskAddresses
                           of the blocks that form the file.
                           It has three parts as noted above.  Notice
                           that all three parts are always there, so
                           that even in a very large file, the first
                           DIRECTSIZE blocks can be located quickly
                           The blocks in the Random index have logical
                           block numbers that are negative.  The logical
                           block number of Indirect[0] is -2 (the FIB is
                           -1) the last possible block's number is
                           -(INDSIZE+DBLINBDSIZE+1)}

                          Direct      : array [0..DIRECTSIZE-1] of DiskAddr;
                          Indirect    : array [0..INDSIZE-1]    of DiskAddr;
                          DblInd      : array [0..DBLINDSIZE-1] of DiskAddr;

                          SegKind     : SpiceSegKind;

                          NumBlksInUse : integer; {segments can have gaps,
                                                   block n may exist when
                                                   block n-1 has never been
                                                   allocated.  NumBlksInUse
                                                   says how many data blocks
                                                   are actually used by the
                                                   segment}
                          LastBlk     : FSBit16;  {Logical Block Number of
                                                   largest block allocated}
```

```
                        LastAddr   : DiskAddr; {DiskAddr of LastBlk }
                        LastNegBlk : FSBit16;  {Logical Block Number of
                                                largest pointer block
                                                allocated}
                        LastNegAddr: DiskAddr  {Block number of
                                                LastNegBlk}
                      );
{5 is the format of a DiskInformationBlock or a PartitionInformationBlock}
                    5: (

                        {The Free List is a chain of free blocks linked
                        by their headers }

                        FreeHead   : DiskAddr; {Hint of Block Number of
                                                the head of the free
                                                list}
                        FreeTail   : DiskAddr; {Hint of Block Number of
                                                the tail of the free
                                                list}
                        NumFree    : FSBit32;  {Hint of how many blocks
                                                are on the free list}
                        RootDirID  : SegID;    {where to find the Root
                                                Directory}
                        BadSegID   : SegID;    {where the bad segment is}

                        {when booting, the boot character is indexed into
                         the following tables to find where code to be
                         boot loaded is found }

                        BootTable  : array [0..25] of DiskAddr; {qcode}
                        InterpTable: array [0..25] of DiskAddr; {micro-
                                                                 code}
                        PartName   : packed array [1..8] of char;
                        PartStart  : DiskAddr;
                        PartEnd    : DiskAddr;
                        SubParts   : array [0..63] of DiskAddr;
                        PartRoot   : DiskAddr;
                        PartKind   : PartitionType;
                        PartDevice : DeviceType
                      );
{6 is the format of a block of a Directory}
                    6: (
                        Entry      : array [0..FILESPERDIRBLK-1] of
                                     DirEntry
                      );
{7 is a format for DiskInformationBlocks which contains a flag denoting
   the meaning of filler words in the header of blocks on the disk;
   it takes advantage of the fact that the definitions of PIBs and DIB are
   intertwined in variant 5 above so that the initial words in a DIB can
   be assumed to contain 0's since initial words are only used in blocks
   which are actually PIBs.  Hence all shugart disk volumes should have
   the value DiskRelative for the FillerKind field by default.}
                    7: (FillerKind : FillerSemantics)
                end;

  ptrDiskBuffer = ^DiskBuffer;
```

```
    Header      = packed record             {format of a block header}
                    SerialNum : DiskAddr; {Actually has the SegID of the
                                           file}
                    LogBlock  : integer;  {logical block number}
                    Filler    : integer;  {holds a hint to a candidate
                                           for the FreeHead}
                    PrevAdr   : DiskAddr; {Disk Address of the next block
                                           in this segment}
                    NextAdr   : DiskAddr; {Disk Address of the previous
                                           block in this segment}
                  end;

    ptrHeader   = ^Header;

    DiskCommand= (DskRead, DskWrite, DskFirstWrite, DskReset, DskHdrRead,
                      DskHdrWrite);  {last ones for error reporting}

var
    DiskSegment : integer;                  {a memory segment for DiskIO}

procedure InitDiskIO;  {initialize DiskIO, called at boot time}

procedure ZeroBuffer(ptr : ptrDiskBuffer);  {write zeroes in all words of
                                             the buffer. When reading an
                                             unallocated block, Zeros are
                                             returned in the buffer}

function WhichDisk(addr : DiskAddr) : integer;  {Tells you which disk
                                                 number a DiskAddr is on}

function AddrToField(addr : DiskAddr) : integer;  {gives you a one word
                                                   short address by taking the
                                                   lower byte of the upper word
                                                   and the upper byte of the
                                                   lower word. The upper byte
                                                   of the upper word can't have
                                                   any significant bits for the
                                                   12 or 24 megabyte disks. The
                                                   lower byte of the lower word
                                                   is always zero (since a disk
                                                   address is a page address,
                                                   which is 256 words
                                                   }

function FieldToAddr(disk: integer; fld : integer) : DiskAddr;
                                                  { Makes a DiskAddr out of
                                                    a short address and a
                                                    disk number
                                                  }
```

```
procedure DiskIO(addr : DiskAddr; ptr : ptrDiskBuffer;
                 hptr : ptrHeader; dskcommand : DiskCommand); {Do a disk
                                                               operation, if
                                                               errors occur,
                                                               exits via
                                                               DiskError}


function LogAddrToPhysAddr(addr : DiskAddr) : DiskAddr;
                                {translate a Logical Disk Address (used
                                 throughout the system) to and from a
                                 physical Disk Address (the kind the disk
                                 contoller sees) Logical Disk Addresses
                                 use a sequential numbering system
                                 Physical Disk Addresses have a
                                 Cylinder-Head-Sector system This routine
                                 calls MapAddr (a private routine which
                                 does the translation) Map Addr
                                 implements interlace algorithm}

function PhysAddrToLogAddr(disk : integer; addr : DiskAddr) : DiskAddr;

function LastDiskAddr(DevType : DeviceType) : DiskAddr; {Gets the Disk
                                                         Address of the last
                                                         possible page on the device}


function NumberPages(DevType : DeviceType) : FSBit32;  {Return the number
                                                        of pages on a device}


procedure DiskReset;  {Reset the disk controller and recalibrate the
                       actuater}

function TryDiskIO(addr : DiskAddr;  ptr : ptrDiskBuffer;
                   hptr : ptrHeader;  dskcommand : DiskCommand;
                   numTries: integer) : boolean;
                                {Try a disk operation, but, return
                                 false if error occurred
                                }

Exception DiskFailure(msg: String; operation: DiskCommand; addr: DiskAddr;
                      softStat: integer);
Exception DiskError(msg: String);
Exception BadDevice;


Var ErrorCnt : Array[IOEFirstError..IOELastError] of integer;
```

```
module DiskParams;


{-------------------------------------------------------------
{
{ Abstract:
{       This program maintains a data base of mappings from disks
{       to their parameters. The format of the data base is:
{
{       1. ! starts a comment terminated by the end of line character.
{       2. Each entry occupies 1 Line.
{    <Disk> <SecPerTrack> <Heads> <NumCylinder> <PreCompCyl> <BootSize>
{
{       Micropolis 16 8 256 128 32
{       This line says that the Micropolis 5.25" drive has 16 sectors per
{       track, 8 heads per cylinder, 256 cylinders, write pre-comp starts
{       at cylinder 128 and the boot size is 32 sectors.
{
{-------------------------------------------------------------}


exports

Function GetParms(DiskName: string;
                  Var Head, Cyls, sectors, precompcyl, bootsize: Integer
                 ): Boolean;

procedure ParamHelp;

Procedure SetUpDiskParams(Automatic: Boolean;
                          DiskName: String;
                      Var NumHeads,
                          NumCylinders,
                          SecPerTrk,
                          BootSize, writecompcyl: Integer);
```

```
module DiskUtility;


{------------------------------------------------------------------------
{
{ DiskUtility -  TV ( Tony Vezza ).
{
{ Copyright (C) 1983, PERQ Systems Corporation
{
{ Abstract:
{       DiskUtility exports procedures to the Vol SubSystem. Contains
{       procedures and functions to perform many disk operations.
{
{------------------------------------------------------------------------}

{$Version V1.0 for POS}

{**************************}  Exports  {**************************}

Imports VolumeSystem From VolumeSystem;
Imports DiskDef From DiskDef;
Imports System From System;


Procedure VInitialize ;

Function  Mount( PID          : PhyDiskID;
                 Labelled     : Boolean ) : VolID;

Procedure DisMount( PID       : PhyDiskID );

Procedure InitDCB( VID        : VolID) ;

Function  CheckVolume( VID    : VolID;
                       DKind  : DiskKinds ) : DiskType;

Function  GetVolName( VID     : VolID) : VolName;

Function  GetDiskSize( VID : VolID) : Long;

Function  FreeDCB( VID        : VolID) : Boolean;

Procedure VolSize( VID        : VolID);

Function  VolToPhyAddr( VA : VolAddress) : Double;

Function  PhyToVolAddr(   VID : VolID;
                          PA  : PhyVolAddress) : VolAddress;

Procedure ToLogHeader(VID     : VolID;
                      PVolHead: PtrVolHeaderBuffer;
                      PLogHead: IOHeadPtr ) ;

Procedure FromLogHeader( VID      : VolID;
                         PLogHead : IOHeadPtr;
                         PVolHead : PtrVolHeaderBuffer ) ;
```

```
Function  PhyToLogAddr( VID : VolID; PA : PhyVolAddress) : LogAddress;

Function  LogToPhyAddr( LA  : LogAddress) : Double;

Function  VolToLogAddr( VA  : VolAddress) : LogAddress;

Function  LogToVolAddr( LA  : LogAddress ) : VolAddress;

Function  FlpyMap(   VA   : VolAddress) : Double;

Function  FlpyUnMap( VID  : VolID;
                     PA   : PhyVolAddress) : VolAddress;

Procedure GetDiskParameters(Var Heads : Integer;
                            Var SectorsPerTrack : Integer;
                            Var NumCylinders: Integer);
```

```
module DoSwap;

    Abstract:

        Turns swapping on or off for the shell.

        Copyright (C) PERQ Systems Corporation, 1982

Version Number V1.1
{////////////////////////////////} EXPORTS {\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\}

Imports CmdParse    from CmdParse;

Procedure DoSwap(args: CString);


Procedure DoSwap(args: CString);

    Abstract:

        Handles the Swap command
```

```
module Dynamic;


Dynamic - Perq dynamic memory allocation and de-allocation.
J. P. Strait      1 Jan 80.
Copyright (C) PERQ Systems Corporation, 1980, 1981, 1982.
```

Abstract:

Dynamic implements Pascal dynamic allocation - New and Dispose. Memory of a given size with a given alignment may be allocated from any data segment with the standard procedure New which calls the NewP procedure of Dynamic.

Data segments are created with CreateSegment from Memory or CreateHeap from Dynamic. Segments created with CreateSegment are automatically enlarged when they become full. They are enlarged by multiples of the segment's increment size until there is enough free memory for the allocation. When an attempt is made to increase the segment past its maximum size, an exception is raised.

Segments created with CreateHeap do not have increment sizes or maximum sizes. Whenever a segment becomes full, another segment of the same size is created and linked to the full segment. This new segment is given the same reference count as the parent. Allocation is potentially done from any of the segments. Thus There are a heap of segments from which to allocate. This heap is identified by the segment number of the first segment allocated. If an allocation is attempted which is larger than the size of the segments, one larger segment is created.

Both heaps and segments may be destroyed by DecRefCount. IncRefCount and DecRefCount applied to a single segment in a heap increment or decrement all segments in the heap. DecIOCount and IncIOCount increment and decrement only a single segment.

Dispose may be used for both segments and heaps. Memory that is deallocated by Dispose becomes a candidate for allocation with New.

The default segment (the one obtained by New(P) without an explicit segment number) is made by CreateSegment(4,4,256) for 1/4 MByte systems and is made by CreateHeap(20) for larger systems. This segment may be destroyed by DecRefCount(0).

Design: Free memory within each segment is linked into a circular freelist in order of address. Each free node is at least two words long and is of the form

```
record Next:   Integer;
Length: Integer;
Rest:   2*Length - 2 words
end;
```

Where Next*2 is the address of the next free node and Length*2 is

the number of free words.

Version Number V2.5

exports

imports Memory from Memory;

```
procedure NewP( S: SegmentNumber; A: integer; var P: MMPointer;
    L: integer );
procedure DisposeP( var P: MMPointer; L: integer );
procedure CreateHeap( var S: SegmentNumber; Size: MMExtSize );
procedure DestroyHeap( S: SegmentNumber );

exception NotAHeap( S: SegmentNumber );

exception SegTooBigForNew( S: SegmentNumber );
```

Abstract:

Raised when allocate out of a segment that has >256 blocks or when try to create a heap of >256 blocks.

Parameters:

The segment allocating from.

```
procedure DisposeP( var P: MMPointer; L: integer );
```

Abstract:

Deallocate memory.

Parameters:
P - Pointer to the memory.
L - Length in words, 0 represents a length of 2**16. If L is odd, L+1 words are de-allocated.

Errors: NilPointer if P is nil.
BadPointer 1) if the Offset part is odd.
2) if Offset+Length > size of segment.
3) if the node to be Dispose overlaps some node that is already free.
4) if the segment is not InUse or not a DataSegment.

```
procedure NewP( S: SegmentNumber; A: integer; var P: MMPointer;
          L: integer );
```

Abstract:

Allocate memory.

Parameters:
S - Number of the segment from which to allocate (if created by CreateSegment) or root segment of the heap (if created by CreateHeap). 0 means the default data segment.
A - Alignment of node in words relative to beginning of segment, 0

represents an alignment of 2**16. if A is odd, A+1 is used as the alignment.
P - Set to point to the memory that was allocated. If the data segment is full and cannot be increased, P is set to nil.
L - Length in words, 0 represents a length of 2**16. If L is odd, L+1 words are allocated.

Errors: FullSegment if the segment has reached its maximum size and there isn't enough room for the node.
FullMemory if NewP tries to expand the segment, but there enough physical memory to do so.
UnusedSegment if S is not InUse.
NotDataSegment if S is not a DataSegment.

```
procedure CreateHeap( var S: SegmentNumber; Size: MMExtSize );
```

Abstract:

Create the root data segment for a Heap.

Parameters:
S - Set to the number of the new segment.
Size - The size of the initial segment and all subsequent segments. Must be <= 256.

Errors: SegTooBigForNew is size is >256.

```
procedure DestroyHeap( S: SegmentNumber );
```

Abstract:

Destroy a Heap or a Segment.

Parameters:

S - The segment number of the root of the Heap.

Errors: UnusedSegment if S is not InUse.
NotDataSegment if S is not a DataSegment.

module Ether10IO;

Abstract:

This module provides the client interface to the 10 Mbaud Ethernet microcode.

Written by: Don Scelza



Version Number V2.5
{**********} Exports {**********}

This module provides the raw I/O interface to the PERQ Systems Ethernet system. The procedures in this module allow the client to send and receive packets on the net.

For details of the Physical and Data Link layers of the network see the document:

The Ethernet
A Local Area Network
Data Link Layer and Physical Layer Specifications

DEC - Intel - XEROX

For details on the PERQ Systems hardware interface to the network see:

Ethernet Interface Programmers Guide

Pradeep Reddy

For details on the interface presented, to this module, by the Ethernet microcode see the file:

Ether10.Micro

Donald A. Scelza

Following is some general information about the client interface presented by the Ethernet microcode and this module:

It is possible to always have a receive pending. If a send command is executed while a receive is pending the internal state of the interface is saved in a register save area in memory. This is done by saving the VA of the DCB for the receive. After the send has completed the receive state is reloaded and the receive is restarted.

In addition to the ability to do a Receive followed by a Send, it is also possible to do multiple Receives. The Receives are linked using the NextDCB field of the Ethernet DCB. When a Receive completes the next Receive in the chain is started.

Command information for the Ethernet driver is provided in an Ethernet

Device Control Block, DCB. All data areas referenced by pointers in the DCB as well as the DCB itself must be LOCKED in memory until the request has completed. They can NOT be moved. The best way to do this is to mark the segment that the buffers are allocated from as UnMovable. This will allow the memory manager to place the buffers in a convient place in memory before they are locked down.

The Ethernet driver needs to have a four (4) word area of memory in which it can save registers. A pointer to this area of memory is provided by the BuffPtr when a Reset command is executed. Once the Reset command has been processed this register save area can NOT be moved. To change the register save area another Reset command must be executed.

The Ethernet DCB must be unmovable while the command is pending.

To wait for the completion of a command it is possible to spin on the Command-In-Progress bit in the status block. This bit will be cleared when the requested command has been completed.

After a receive the Bits field of the status block has the number of bits that were received. To translate this into the number of data bytes you must perform a number of operations. First divide it by 8. This will give the number of bytes that were received. If the number is not evenly divisable by 8 then there was a transmition error. After the division you must subtract off the number of bytes in the header and the CRC. There is a total of 18 bytes in these two portions of the packet.

The Ethernet controler can receive packets that are addressed in the following ways:

a) Packets addressed to this machine.
b) Packets addressed to any machine.
c) Packets addressed to five of 256 groups.
d) Packets addressed to any group.

The Reset command is used to set up the addressing for a given machine.

```
imports SystemDefs from SystemDefs;
imports System from System;

{
{ Define the types and variables used by the Network stuff.
{ }

type

    {
    { These are the valid commands for the Ethernet interface.
    { }

    EtherCommand = (EReset, EReceive, EPromiscuousReceive, ESend);

    {
    { An Ethernet address is 48 bits long.  It is made up of 6
    { octets or in our case 3 words.
    { }

    EtherAddress = packed record   { An address on the net is 48 bits }
        High: integer;
        Mid:  integer;
        Low:  integer;
        end;

    {
    { This record defines an Ethernet status block.  The first
    { 15 bits of the block are defined by the hardware interface.
    { The 16th bit of the first word and the second word are defined by the
    { Ethernet microcode.
    {
    { Alignment:    Double word.
    { Locked:       Yes.
    { }

    EtherStatus = packed record      { The status record. }
        CRCError:       boolean;     { There was a CRC error.}
        Collision:      boolean;     { There was a collision }
        RecvTrans:      boolean;     { 0 - receive has finished.  1 for
                                       trans. }
        Busy:           boolean;     { The interface is bust. }
        UnUsed4:        boolean;
        ClockOver:      boolean;     { The microsecond clock overflowed. }
        PIP:            boolean;     { There is a Packet In Progress. }
        Carrier:        boolean;     { There is traffic on the net. }
        RetryTime:      0..15;
        UnUsed12:       boolean;
        UnUsed13:       boolean;
        SendError:      boolean;     { Could not send packet after 16 tries.}
        CmdInProgress:  boolean;     { There is a command pending. }
        BitsRecv:       integer;     { Number of bits that were received. }
        end;

    {
    { This record defines the header for an Ethernet transfer.
```

```
{
{ Alignment:    8 word.
{ Locked:       Yes.
{ }

EtherHeader = packed record
    UnUsed:     Integer;        { A filler word.  This must be here.}
    Dest:       EtherAddress;
    Src:        EtherAddress;
    EType:      Integer;        { Type field defined by XEROX }
    end;

{
{ This record provides the definition of an EtherBuffer.
{
{ Alignment:    1k word.
{ Locked:       Yes.
{ }

EtherBuffer = array [0..749] of integer;

{
{ Define all of the pointers that we need.
{ }

pEtherStatus = ^EtherStatus;
pEtherBuffer = ^EtherBuffer;
pEtherHeader = ^EtherHeader;
pEtherDCB = ^EtherDCB;

{
{ This is the definition of an Ethernet Device Control Block.
{
{ Alignment:    Quad word.
{ Locked:       Yes.
{ }


EtherDCB = packed record
    HeadPtr:    pEtherHeader;
    BuffPtr:    pEtherBuffer;
    StatPtr:    pEtherStatus;
    Cmd:        EtherCommand;
    BitCnt:     Integer;            { Total bits in buffer and header }
    NextDCB:    pEtherDCB;
    end;

{
{ This is the definition that is used to create the register save
{ area.  Must exist across transfers.
{
{ Alignment:    Double word.
{ Locked:       Yes.
{ }

EtherRegSave = record
```

```
        RecvDCB:    pEtherDCB;
        SendDcb:    pEtherDCB;
        end;

    pEtherRegSave = ^EtherRegSave;


    {
    { This is the definition of the structure that is used to set
    { the physical address of this machine and the groups that we are
    { to look for.
    {
    { Alignment:    1 word.
    { Locked:       Yes, during Reset.
    { }

    EtherAdRec = packed record
        LowAddress: Integer;        { Low word of Physical address }
        MCB:        0..255;         { Mulitcast command byte. }
        MultCst1:   0..255;         { Five group addresses. }
        MultCst2:   0..255;
        MultCst3:   0..255;
        MultCst4:   0..255;
        MultCst5:   0..255;
        end;

    pEtherAdRec = ^EtherAdRec;

    {
    { Following are the definitions that are used to deal with the
    { micro-second clock.
    {
    { The microsecond clock takes a two word combined control and
    { status block.  The first word of the block gives the number
    { of microseconds to be loaded into the clock.
    { The second word provides the status information from the
    { clock.  Once a clock command has been started it is
    { possible to spin on the CmdInProgress bit in the control
    { block.  When the bit is cleared the specified number of
    { micro-seconds has elapsed.
    {
    { Alignment:    Double word.
    { Locked:       Yes.
    { }

type

    uSClkDCB = packed record
        uSeconds: integer;
        UnUsed0:  boolean;
        UnUsed1:  boolean;
        UnUsed2:  boolean;
        UnUsed3:  boolean;
        UnUsed4:  boolean;
        UnUsed5:  boolean;
        UnUsed6:  boolean;
```

```
        UnUsed7:    boolean;
        UnUsed8:    boolean;
        UnUsed9:    boolean;
        UnUsed10:   boolean;
        UnUsed11:   boolean;
        UnUsed12:   boolean;
        UnUsed13:   boolean;
        UnUsed14:   boolean;
        CmdInProg:  boolean;
        end;

    puSC1kDCB = ^uSC1kDCB;

{
{ Define the constants for the address block supplied to PERQ Systems by
{ Xerox.
{
{ High 16 bits (2 octets) are 02 1C (Hex).
{ Next 8 bits (1 octet) is 7C (Hex).
{
{ The low order byte of the second PERQ word as well as the third PERQ word
{ are PERQ Systems defined.  Currently the low order byte of the second word
{ is used to define the type of interface.  The valid values are:
{
{       0 - Interface is on an IO option board.
{       1 - Interface is on the IO board
{ }

const

    TRCCAdrMid = 31744;         { 7C hex in the high order 8 bits. }
    TRCCAdrHigh = 540;          { 02 1C hex. }
    EBoardOption = 0;           { The interface is on an I/O Option board.}
    EBoardIO = 1;               { The interface is on the I/O board. }

{
{ These are some other useful constants.
{ }

const

    MinDataBytes = 46;          { Smallest number of data bytes in a
                                  packet.}
    MaxDataBytes = 1500;        { Largest number of data bytes in a packet.}
    NumDCBs = 16;               { The number of DCBs, commands, possible at}
                                { a single time }

{
{ Define the constants for the multicast command byte.
{ }

const
    MltCstAll = 0;              { Receive all multicasts. }
    MltCstNone = #377;          { Don't receive any multicast packets. }
    MltCstAddr = #376;          { Return the Physical addr of this device. }
    MltCstGrp = 1;              { Only receive specified groups. }
```

```
{
{ These are the procedures exported by this module.
{ }

procedure E10Init;
procedure E10IO(Cmd: EtherCommand;  Header: pEtherHeader;  Buff:
pEtherBuffer;
                Stat: pEtherStatus;  Bytes: Integer);
procedure E10Wait(Stat: pEtherStatus);
procedure E10Reset(Ptr: pEtherAdRec);
function  E10DataBytes(RecvBits: Integer): Integer;
function  E10GetAdr: EtherAddress;
procedure E10State(var NumSend, NumReceive: Integer);
procedure E10WIO(Cmd: EtherCommand;  Header: pEtherHeader;
                Buff: pEtherBuffer;  Stat: pEtherStatus;  Bytes: Integer);

{
{ These are the exceptions that may be raised by this module.
{ }

exception E10NInited;

     Abstract:

          This exception will be raised if any procedures in this package
          are called before E10Init.

exception E10NReset;

     Abstract:

          This exception will be raised if any transfer commands are
          executed before a E10Reset is done.

exception E10ByteCount;

     Abstract:

          This exception is raised if a byte count passed to this interface
          is not in the valid range.  The number of data bytes in an
          Ethernet packet must be in the range 46 <-> 1500, (MinDataBytes
          <-> MaxDataBytes).

exception E10DByteError;

     Abstract:

          This exception is raised if the number of Bits passed to
          E10DataBytes does not form a valid packet.
```

```
exception E10BadCommand;
```

Abstract:

This exception is raised if a bad command is given to any of the routines in this package.

```
exception E10TooMany;
```

Abstract:

This exception is raised if more than NumDCBs commands are executed at any time.

```
exception E10STooMany;
```

Abstract:

This exception is raised if more the client tries to execute more than one send.

```
exception E10ReceiveDone(Stat: pEtherStatus);
```

Abstract:

This exception is raised when a receive command has finished. It is raised by the Pascal level interrupt routine for the net. The exception is only raised when the ethernet exception for data interrupts has been turned on. The following does the trick:

```
Import IO_Unit From IO_Unit; Setting := True;
IOSetExceptions(Ether10, IODataInterrupt, Setting);
```

Note that process clean up resets the bit on exit of a program.

Parameters:

Stat will be set to the status pointer of the command that finished.

```
exception E10NoHardware;
```

Abstract:

This exception is raised by E10GetAdr if there is no ethernet board in the machine.

procedure E10Init;

Abstract:

This procedure is used to intialize the Ethernet module. It must be called before any other procedure in this package are used.

This procedure is called ONCE at boot time by the O.S. It MUST NOT be called by user programs.

Side Effects: This procedure will allocate any memory used by this module.

procedure E10IO(Cmd: EtherCommand; Header: pEtherHeader; Buff: pEtherBuffer; Stat: pEtherStatus; Bytes: Integer);

Abstract:

This procedure is used to start an Ethernet I/O operation and return.

Parameters:

Cmd is the command that is to be executed.

Header is a pointer to an Ethernet header block. The client must fill in all fields of this header.

Buff is a pointer to the buffer that is to be sent or filled.

Stat is a pointer to a status block for use during this command.

Bytes is the number of data bytes that are to be transfered. This value must be between 46 and 1500

Exceptions:

E10NInited: Raised if this procedure is called before EtherInit.

E10NReset: Raised if this procedure is called before EReset.

E10ByteCount: Raised if Bytes is not in the valid range.

E10BadCommand: This is raised if the command passed is not Send, Receive or PromisciousReceive.

E10TooMany: is raised if too many commands are executed at a given time.

E10STooMany: is raised if more than one send command is executed.

procedure E10Wait(Stat: pEtherStatus);

Abstract:

Waits for the completion of some Ethernet request.

Parameters:

Stat is the pointer to the EtherStatus that was provided when the command was initiated.

Exceptions: E10NInited: Raised if this procedure is called before E10Init.

E10NReset: Raised if this procedure is called before EReset.

procedure E10Reset(Ptr: pEtherAdRec);

Abstract:

This procedure is used to reset the Ethernet interface.

Parameters:

Ptr is a pointer to the address record that is to be used for the reset.

Exceptions: E10NInited: Raised if this procedure is called before EtherInit.

function E10DataBytes(RecvBits: Integer): Integer;

Abstract:

This procedure is used to obtain the number of data bytes that are in a packet that was received over the network.

Parameters:

RecvBits is the number of bits that were in the packet. This value will come from the BitsRecv field of the status block.

Results: This function will return the number of data bytes that were in the packet.

Exceptions:
E10NInited: Raised if this procedure is called before EtherInit.

E10NReset: Raised if this procedure is called before EReset.

E10DByteError: Raised if the numebr of bits in the packet was not a multiple of 8 or if the number of data bytes was less than MinDataBytes.

```
function  E10GetAdr: EtherAddress;
```

Abstract:

This function will return the address of this machine.

Exceptions:
E10NInited: Raised if this procedure is called before E10Init.

E10NoHardware: Raised if there is no ethernet board in the machine.

```
procedure E10State(var NumSend, NumReceive: integer);
```

Abstract:

This procedure is used to return the internal state of the Ethernet interface.

Parameters:

NumSend will be set to the number of Sends that are pending.

NumReceive will be set to the number of receives that are pending.

Exceptions:
E10NInited: Raised if this procedure is called before E10Init.

E10NReset: Raised if this procedure is called before EReset.

```
procedure E10WIO(Cmd: EtherCommand;  Header: pEtherHeader;
          Buff: pEtherBuffer;  Stat: pEtherStatus;  Bytes: Integer);
```

Abstract:

Starts an Ethernet I/O operation and waits for it to complete.

Parameters:

Cmd is the command that is to be executed.

Header is a pointer to an Ethernet header block. The client must fill in all fields of this header.

Buff is a pointer to the buffer that is to be sent or filled.

Stat is a pointer to a status block for use during this command.

Bytes is the number of data bytes that are to be transfered. This value must be between 46 and 1500

Exceptions: E10NInited: Raised if this procedure is called before E10Init.

E10NReset: Raised if this procedure is called before EReset.

E10ByteCount: Raised if Bytes is not in the valid range.

E10BadCommand: This is raised if the command passed is not

Send, Receive or PromisciousReceive.

E10TooMany: is raised if too many commands are executed at a given time.

E10STooMany: is raised if more than one send command is executed.

```
module EtherInterrupt;

    Abstract:

        This module provides the interrupt service for the 10 MBaud
        ethernet.

        Written by: Don Scelza.

        Copyright (C) PERQ Systems Corporation, 1981

        {********************} Exports {********************}

        imports Ether10IO from Ether10IO;

        var
            StackPointer: Integer;
            DCBStack: array[1..NumDCBs] of pEtherDCB;
            RListHead, RListTail, SListHead: pEtherDCB;
            SendsPosted, RecvsPosted: Integer;

        function PopDCB: pEtherDCB;
        procedure PushDCB(Ptr: pEtherDCB);
        procedure E10Srv;

function PopDCB: pEtherDCB;

    Abstract:

        Get the next free DCB from the stack.

    Results: Return a pointer to the next free DCB.

    Side Effects: Move the stack pointer.

procedure PushDCB(Ptr: pEtherDCB);

    Abstract:

        Push a free DCB onto the DCB stack.

    Parameters:

        Ptr is a pointer to the DCB that is to be pushed onto the stack.

    Side Effects: Move the stack pointer.
```

```
procedure E10Srv;

    Abstract:

        This is the 10 megabuad Ethernet interrupt routine.

    Exceptions: This procedure raises E10ReceiveDone if a receive
        completes.
```

```
module EtherTime;

Copyright (C), 1982, 1983 PERQ Systems Corporation.

Written by:  Mark G. Faust

Version Number V1.2
          {**************} exports {**************}

function GetEtherTime(TimeOut :integer) :string;

function GetEtherTime(TimeOut   :integer) :string;
```

Abstract:

Get the current time from an EtherNet time server. Return a Perq time string in standard format. Time out after specified number of jiffies. If we time out then we return the null string.

```
module Except;

Except - Perq Pascal Exception Routines.

J. P. Strait   10 Dec 80.

Copyright (C) PERQ Systems Corporation, 1980, 1981, 1982, 1983.
```

Abstract:

Module Except provides the following things:

1) Definitions of the microcode generated exceptions.

2) A procedure to tell the microcode which segment number these exceptions are defined in.

3) The default handler of all exceptions. The compiler enables this handler in every main program.

4) A Pascal routine to search the stack in when an exception is raised.

Design: The file Except.Dfs is included into Perq.Micro as well as into this module. It defines routine numbers for the exceptions generated by the microcode. Note that there must be agreement between these constants and the routine numbers of the exception definitions. No program checks these--if you add or remove exception definitions you must be sure to update Except.Dfs in the appropriate way.

The routine number of RaiseP is also defined in Except.Dfs as 0. Since the microcode must know this, it is strongly suggested that it not be modified.

The routine number of InitExceptions is not needed by the compiler or Perq.Micro, but it has been assigned routine number 1 so that its number will not change when new exceptions are defined. This means that new exceptions may be defined without requiring that the operating system be re-linked.

```
Version Number V3.1

exports

const ExceptVersion = '3.1';

procedure RaiseP( ES, ER, PStart, PEnd: Integer );
procedure InitExceptions;
```

```
exception Abort( Message: String );
exception Dump( Message: String );

exception XSegmentFault( S1,S2,S3,S4: Integer );  { segment fault }

exception XStackOverflow;                         { stack overflow }

exception DivZero;   { division by zero }

exception MulOvfl;   { overflow in multiplication }

exception StrIndx;   { string index out of range }

exception StrLong;   { string to be assigned is too long }

exception InxCase;   { array index or case expression out of range }

exception STLATETooDeep; { parameter in STLATE instruction is too large }

exception UndfQcd;   { execution of an undefined Q-code }

exception UndfInt;   { undefined device interrupt detected }

exception IOSFlt;    { segment fault detected during I/O }

exception MParity;   { memory parity error }

exception EStack;    { E-stack wasn't empty at INCDDS }

exception OvflLI;    { Overflow in conversion to integer from Long Integer }

exception OverReal;  { floating point overflow }

exception UndeReal;  { floating point underflow }

exception RealDivO;  { floating point division by zero }

exception Real2Int;  { floating point real to integer overflow }

exception UnImplQCode; { QCode is defined, but not implemented in this
                                 interpreter }

    var ExcSeg: Integer;

procedure InitExceptions;

    Abstract:

        InitExceptions tells the microcode what segment number to use when
        raising its own exceptions.  The segment number is the one that
        the system assigns to this module.

        Side affects: ExcSeg is set to the current segment number. The
        current segment is kept resident.
```

procedure RaiseP( ES, ER, PStart, PEnd: Integer );

Abstract:

RaiseP is called to raise an exception. The compiler generates a call to RaiseP in response to

```
raise SomeException( original parameters )
```

in the following way

```
Push original parameters onto the MStack.
RAISE SegmentNumber(SomeException) RoutineNumber(SomeException)
ParameterSize
```

The microcode calls RaiseP in the following way:

```
Push parameters onto the MStack if appropriate.
ParameterSize := WordsOfParameters.
Error := ErrorNumber, Goto(CallRaise).
```

where CallRaise does the following:

```
SaveTP := TP.
Push ExcSeg onto the MStack.
Push Error onto the MStack.
Push SaveTP-ParameterSize+1 onto the MStack.
Push SaveTP+1 onto the MStack.
call RaiseP.
```

Parameters:

ER - Routine number of the exception to be raised.
ES - Segment number of the exception to be raised.
PStart- Pointer to the original parameters (as an offset from the base of the stack).
PEnd - Pointer to the first word after the original parameters (as an offset from the base of the stack).

Calls: Appropriate exception handler or HandleAll.

Design: See the "PERQ QCode Reference Manual, Q-Machine Architecture" for a description of the format of exception enable blocks and the format of variable routine descriptors.

RaiseP searches the exception enable list of each routine in the dynamic chain. When it finds one that matches ER and ES it searches the dynamic chain again to see if the specified handler is already active. If it is active, RaiseP continues searching the exception lists and dynamic chain where it left off. This is done in order to allow a handler to re-raise the same exception, and to prevent unlimited recursion in an exception handler that has a bug.

If an exception handler is found, the original parameters are pushed onto the MStack and the handler is called with CALLV.

If no exception handler is found, HandleAll is called.

*** RaiseP may not contain any exception handlers.

*** RaiseP must be guaranteed to be resident.

```
module FileAccess;
```

Abstract:

Module to handle reading, writing, entering and deleting files independant from the directory structure.

Written by the CMU Spice Group

```
Version Number V1.8
{**********} exports {**************************}

imports Arith from Arith;
imports DiskIO from DiskIO;
imports AllocDisk from AllocDisk;

function  CreateSpiceSegment(partition : integer; kind : SpiceSegKind) :
    SegID;
procedure DestroySpiceSegment(id : SegID);
procedure TruncateSpiceSegment(id : SegID; len : integer);
procedure ReadSpiceSegment(id : SegID; firstblk,numblks : integer;
                           ptr : ptrDiskBuffer);
procedure WriteSpiceSegment(id : SegID; firstblk,numblks : integer;
                            ptr : ptrDiskBuffer);
procedure Index(logblk : integer; var indblk,indoff : integer);

Exception BadLength(len: integer);
```

Abstract:

Raised if try to truncate file to a length < 0

Parameters:

len is bad length

```
Exception NotAFile(id: SegID);
```

Abstract:

Raised when an operation is attempted and the SegID passed does not seem to be the id for a valid file

Parameters:

id is the bad id

```
Procedure Index(logblk : integer; var indblk,indoff : integer);
```

Abstract:

Find the index block and the offset from the top of the block for a logical block of a file

Parameters:
logBlk - the logical block of the file to look up; may be negative
indBlk - the logical block number of the index block which holds the address for logblk
indoff - the offsetin indBlk to use in reading the address (the array index to use in DiskBuffer^.Addr). It is correctly set even if the indBlk is the FIBlk

```
function  CreateSpiceSegment(partition : integer;
        kind : SpiceSegKind) : SegID;
```

Abstract:

Create a new empty file on partition specified

Parameters:

partition is the partition in which to allocate file; kind is the type of segment

Returns: ID of file created

Errors: Raises NotAFile if block at id is not a valid FIBlk

```
Procedure DestroySpiceSegment(id : SegID);
```

Abstract:

Delete a file

Parameters:
id is the SegId of file to delete

SideEffects: removes id from filesystem

Errors: Raises NotAFile if block at id does not seem to be a valid FIBlk

```
Procedure TruncateSpiceSegment(id : SegID; len : integer);
```

Abstract:

Removes blocks from file to make the new length len

Parameters:

id is the SegId of file; len is the new length (one greater than the last logical block number since files start at 0)

SideEffects: Shortens the file

Errors: Raises BadLength is length to truncate file to is < 0 Raises NotAFile if block at id does not seem to be a valid FIBlk

```
Procedure ReadSpiceSegment(id : SegID; firstblk,numblks : integer;
          ptr : ptrDiskBuffer);
```

Abstract:

Reads one or more blocks from file

Parameters:

```
id       - the SegId of file;
firstBlk - the logical blk # of first to read
numBlks  - the number of blocks to read
ptr      - where the data should be put NOTE: If the blocks
     specified to read don't exist; ptr^ is filled with zeros
```

Errors: Raises NotAFile if block at id is not a valid FIBlk

```
Procedure WriteSpiceSegment(id : SegID; firstblk,numblks : integer;
          ptr : ptrDiskBuffer);
```

Abstract:

Writes one or more blocks onto file

Parameters:

```
id       - the SegId of file;
firstBlk - the logical blk # of first to write
numBlks  - the number of blocks to write
ptr      - where the data should come from
```

SideEffects: Changes the data in the file and may cause new blocks to be allocated and file length changed

Errors: Raises NotAFile if block at id is not a valid FIBlk

```
module FileDefs;

     Abstract:


          Defines some constants and types needed by various people so
          FileSystem doesn't need to import DiskIO in its export section

          Written by: Brad A. Myers  3-Mar-81
          Copyright (C) 1981 PERQ Systems Corporation
Version Number V1.2


exports

Imports GetTimeStamp from GetTimeStamp; {Using TimeStamp}

const
  DBLZERO             = nil;     {a two word 0}

type
  FSBit8              = 0..255;
  FSBit16             = integer;
  FSBit32             = ^integer; {will be a long when compiler knows about
                                   them}

Const DISKBUFSIZE  = 256;       {defined by hardware, 256 words per sec}

type SegID          = FSBit32;  {In SpiceSeg, the virtual address of the
                                 -1 block of a file}
     DiskAddr       = FSBit32;  {The virtual address of a DiskBlock}

     SimpleName     = string[25];   {only the filename in the directory}
     PathName       = string[100];  {full name of file with partition and
                                     dev}
     PartialPathName = string[80]; {file name including all directories}
     FSOpenType     = (FSNotOpen, FSOpenRead, FSOpenWrite, FSOpenExecute);
     FSDataEntry    = packed record
                        FileBlocks       : integer;  {Size of file in blocks}
                        FileBits         : 0..4096;  {Number of bits in last
                                                      blk}
                        FileSparse       : Boolean;  {true if can be sparse}
                        FileOpenHow      : FSOpenType;  {howOpen}
                        FileCreateDate   : TimeStamp;
                        FileWriteDate    : TimeStamp;
                        FileAccessDate   : TimeStamp;
                        FileType         : integer;  {see FileType.pas}
                        FileRights       : integer;  {protection code}
                        FileOwner        : FSBit8;   {UserId of file owner}
                        FileGroup        : FSBit8;   {GroupId}
                        Filename         : PartialPathName;
                   end;
   ptrFSDataEntry = ^FSDataEntry;
```

```
module FileDir;

    Abstract:

        The directory structure for PERQ FileSystem

        Written by:  CMU Spice Group

Version Number V2.6
{**********************}Exports{**********************************}

imports FileDefs from FileDefs;

function  GetFileID(name : PathName) : SegID;
function  PutFileID(var name : PathName; id : SegID) : boolean;
function  DeleteFileID(name : PathName) : SegID;
function  GetDisk(var name : PathName; var partition : integer) : boolean;

var
  DefaultPartitionName : SimpleName; {includes device name and ends in
                                      a ">" }
  DefaultDeviceName    : SimpleName; {ends in a colon}

Function GetDisk(var name : PathName; var partition : integer) : boolean;

    Abstract:

        Given a name, remove the device and partition specification and
        find the partition number

    Parameters:
        name - the full file name to parse; the device and partition are
            optional.  The device and partition if there are removed from
            the name string;

        partition - set to the partition specified or the default

    Returns: False if specified device or partition malformed or not there

    SideEffects: Mounts the partition if not already

    Calls: FindPartition, MountPartition

Function GetFileID(name : PathName) : SegID;

    Abstract:

        Find the SegID for name (does a lookUp)

    Parameters:
        name - the full name (including all directories and optional
            device and partition) of the file to look up
```

Returns: The SegID of the file or DBLZERO if not there or mal-formed

Calls: ParseFilename, GetRootDirID, GetIDFromDir

Function PutFileID(var name : PathName; id : SegID) : boolean;

Abstract:

enters name with SegID id into a directory

Parameters:

name - the full name (including all directories and optional device and partition) of the file to enter; it is changed to remove all ">..>" and ">.>"s and remove the device (the name returned can be entered in the FileID block's FSData.Filename).

id - SegID of file;

Returns: True if file successfully entered; false if device, partition or a sub-directory is mal-formed NOTE: ***IT IS ILLEGAL TO CALL PutFileID FOR A NAME THAT IS ALREADY IN THE*** ***DIRECTORY BUT THIS IS ONLY SOMETIMES CAUGHT IF ATTEMPTED***

Calls: ParseFilename, GetRootDirID, GetIDFromDir, PutIDInDir

Function DeleteFileID(name : PathName) : SegID;

Abstract:

Removes the directory entry for name

Parameters:

name - the full name (including all directories and optional device and partition) of the file to remove from directory

Returns: SegID of file removed from Directory or DBLZERO if not there or part of name is mal-formed

Calls: ParseFilename, GetRootDirID, GetIDFromDir

```
module FileSystem;

     Abstract:


          Spice Interim File System.

          Written by: Richard F. Rashid
          February 24, 1981
          Copyright (C) 1981 - Carnegie-Mellon University
Version Number V7.4

{********************} Exports {********************}

imports FileDefs from FileDefs;

const
  FSVersion     = '7.3';    { File system version number }
  BlksPerFile   =#077777;   { Max blocks in each file }
  FirstBlk      =0;         { Block number of the first data block }
                            { in a file }
  LastBlk       =#077776;   { Block number of the last data block }
                            { in a file. }
  FIBlk         =-1;        { Block number of the File Information Block }
  BootLength    = 60 + 128; { Size of the bootstrap area on disk--the }
                            { first n blocks on the disk.  the microcode }
                            { boot area is 60 blocks, the Pascal boot  }
                            { area is 128 blocks (32K). }
  StartBlk      =BootLength;{ The block number of the FIBlk of the first}
                            { user file. }
  SysFile       = -1;       { File ID of the system area on disk. }
  SEARCHSIZELIST = 5;       { Max number of directories on search list. }

type
  DirBlk= Record                        { Record for reading disk blocks }
           Case Integer Of
           2: (
                Buffer:Array[0..255] Of Integer
              );
           3: (
                ByteBuffer: Packed Array [0..511] of FSBit8
              )
           End;

  PDirBlk= ^DirBlk;

  FileID        = integer;
  BlkNumbers    = integer;

  SearchList    = array[1..SEARCHSIZELIST] of PathName;
  ptrSearchList = ^SearchList;

var
  FSDirPrefix:PathName;  {current default directory including device and
                          part}
  FSSysSearchList: SearchList;
```

```
function FSLookUp(FileName:PathName;Var BlkInFile,BitsInLBlk: Integer):
    FileID;                                  {uses current system search list}

function FSLocalLookUp(FileName:PathName; Var BlkInFile,BitsInLBlk:
    Integer): FileID;    {doesn't use any search lists}

function  FSSearch(var slist : SearchList; var FileName : PathName;
                   var BlkInFile, BitsInLBlk: integer) : FileID;
                       {uses specified search list instead of system one; is
                        var so no copying; changes FileName to be full
                        filename actually used}
function  FSEnter(FileName:PathName): FileID;
procedure FSClose(UserFile:FileID; Blks,Bits:Integer);
procedure FSBlkRead(UserFile:FileID; Block:BlkNumbers; Buff:PDirBlk);
procedure FSBlkWrite(UserFile:FileID; Block:BlkNumbers; Buff:PDirBlk);
procedure FSInit;
procedure FSMount(disk : integer);
procedure FSDismount(disk : integer);
procedure FSSetPrefix(prefixname : PathName);  {FSSetPrefix just assigns
                                                the vble; use
                                                FileUtils.FSSetPath to do
                                                processing on new path}
procedure FSGetPrefix(var prefixname : PathName);
function  FileIDtoSegID(id : FileID) : SegID;
function  SegIDtoFileID(id : SegID) : FileID;
procedure FSSetupSystem(bootchar: integer);
procedure FixFilename(var filename : PathName; nulliserror : boolean);
Function FSIsFSDev(name: PathName; var devName: String): integer;

Exception FSNotFnd(name: PathName);
```

Abstract:

Raised if file looked up is not found. If this exception is not handled by client, the lookup or search will return zero

Parameters:

name is the name not found

```
Exception FSBadName(name: PathName);
```

Abstract:

Raised if file entered is illegal because: 1) the device or partition specified is not valid 2) a directory name specified does not exist 3) the length of the simpleName is > 25 characters If this exception is not handled by the client, the Enter will return zero

Parameters:

name is the name that is illegal

```
Function FSInternalLookUp(FileName:PathName; Var
```

```
        BlkInFile,BitsInLBlk:Integer):
                        FileID;

Exception FSDirClose;
```

Abstract:

Raised if attempt to FSClose a directory file. This is usually a bad idea since directories are spare files with an invalid length field.

RESUME: Allowed. Will close the file as if nothing had happened.

```
        const
          FSDebug = false;

Function SegIDtoFileID(id : SegID) : FileID;
```

Abstract:

Convert a two word SegId into a one word fileID

Parameters:

id is a two word segID

Returns: A one word FileID; it may be pos or neg or zero

```
Function FileIDtoSegID(id : FileID) : SegID;
```

Abstract:

Convert a one word FileID into a two word SegID

Parameters:

id is a one word FileID

Returns: a two word SegID

```
Procedure FSInit;
```

Abstract:

Initializes the FileSystem; call BEFORE FSSetUpSystem; Also initialize SegSystem and DirSystem

SideEffects: Initializes; sets global Initialized to true; sets Prefix and Search list to null

```
Procedure FixFilename(var filename : PathName; nulliserror : boolean);

    Abstract:

        Makes fileName a full path name by adding as many defaults as
        necessary

    Parameters:

        filename is name to fix; it is modified to have the full path name
        as follows:
        (dev):(rest) - no change
        :(rest)      - adds DefaultDevice from AllocDisk to front
        >(rest)      - adds DefaultPartition from AllocDisk to front
        (rest)       - adds FSDirPrefix to front; if nullIsError-then no
             change to name if fileName = '' else changes '' to
             FSDirPrefix

    Errors: allows STRLong to pass through from PERQ_String; This means
        that fileName is invalid

Procedure FSMount(disk: integer);

    Abstract:

        Mounts the disk specified and prints all partitions

    Parameters:

        Disk is device to mount (0=HardDisk, 1=Floppy)

    Calls: DeviceMount and DisplayPartitions

Procedure FSDismount(disk: integer);

    Abstract:

        Dismounts the disk specified and prints all partitions

    Parameters:

        Disk is device to dismount (0=HardDisk, 1=Floppy)

    Calls: DeviceDismount and DisplayPartitions

Procedure FSSetPrefix(prefixname : PathName);

    Abstract:

        Sets the default pathName
```

Parameters:

prefixname is new name; no checking is done

SideEffects: changes FSDirPrefix

```
Procedure FSGetPrefix(var prefixname : PathName);
```

Abstract:

Returns the default pathName

Parameters:

prefixname is current name; it is set with current value

```
Function FSInternalLookUp(FileName:PathName;Var
                          BlkInFile,BitsInLBlk:Integer):FileID;
```

Abstract:

Does a lookup of FileName in the current path only.

Parameters:

FileName is a filename. BlkInFile and BitsInLBlk are set with the number of blocks in the file and the number of bits in the last block respectively.

Returns: 0 if file doesn't exist; else the FileID of the file

Errors: This procedure does not raise any errors

SideEffects: Sets the FileAccessDate of the file

Calls: FixFileName, GetFileID, GetTStamp, SegIDToFileID

```
Function FSLocalLookUp(FileName:PathName; Var BlkInFile,
        BitsInLBlk:Integer): FileID;
```

Abstract:

Does a lookup of FileName in the current path only.

Parameters:

FileName is a filename. BlkInFile and BitsInLBlk are set with the number of blocks in the file and the number of bits in the last block respectively.

Returns: 0 if file doesn't exist; else the FileID of the file

SideEffects: Sets the FileAccessDate of the file

Errors: Raises FSNotFnd if file not there (if not caught, then lookup returns 0)

Calls: InternalLookUp

Function FSSearch(var slist : SearchList; var filename : PathName; var blkinfile,bitsinlblk: integer) : FileID;

Abstract:

Does a lookup of FileName straight first and then with each of the names in slist on the front.

Parameters:

slist - a searchList; any non-'' entries are assumed to be paths and are put on the front of the filename. The first one to be tried is slist[1]. The first match is the one used; No checking is done on the validity of the entries in slist

filename - the file to be looked up; it is changed to be the full name of the file if found. If fileName is empty then file not found.

BlkInFile and BitsInLBlk - set with the number of blocks in the file and the number of bits in the last block respectively.

Returns: 0 if file doesn't exist in any path; else the FileID of the file

SideEffects: Sets the FileAccessDate of the file

Errors: Raises FSNotFnd if file not there (if not caught, then lookup returns 0)

Calls: FSLocalLookUp, Concat

Function FSLookUp(FileName: PathName; Var BlkInFile, BitsInLBlk: Integer): FileID;

Abstract:

Does a lookup of fileName first in the current path and then in each of the entries of the system search list.

Parameters:

filename is the file to be looked up; BlkInFile and BitsInLBlk are set with the number of blocks in the file and the number of bits in the last block respectively.

Returns: 0 if file doesn't exist in any path; else the FileID of the file

SideEffects: Sets the FileAccessDate of the file

Errors: Raises FSNotFnd if file not there (if not caught, then lookup returns 0)

Calls: FSSearch with FSSysSearchList as the sList

```
Function FSEnter(FileName:PathName): FileID;
```

Abstract:

Enters the file in the current path.

Parameters:

filename is the file to be entered. It may or may not exist; if not exists then is created;

Returns: 0 if file can't be created because part of its name is invalid (e.g. the device, partition or directory specified doesn't exist) else the FileID of the file

SideEffects: Creates a file if necessary and enters it into the directory; if creating, then sets size to zero and create date; Sets type to 0 (UnknownFile); whether or not creating; sets WriteDate and AccessDate

Errors: Raises FSBadName if name passed is illegal due to a device, partition, or directory name in path not existing or target name is longer than 25 characters or illegal in some other way. If this exception is not caught, Enter returns zero.

```
Procedure FSClose(UserFile:FileID; Blks,Bits:Integer);
```

Abstract:

Closes a file (setting size).

Parameters:

UserFile is ID of file to close; Blks is the size of the file in blks and bits is the number of bits in the last block;

SideEffects: Truncates file to size specified; does a FlushAll

Errors: Raises FSDirClose if attempt to close a directory file. If resume from this exception, then closes normally.

Procedure FSBlkWrite(UserFile:FileID; Block:BlkNumbers; Buff:PDirBlk);

Abstract:

Writes one block onto a file.

Parameters:

UserFile is ID of file to write on Block is number of block to write (starting at zero); Buff is buffer holding data to write onto the file at Block

SideEffects: Changes the data of block Block

Calls: WriteSpiceSegment

Procedure FSBlkRead(UserFile:FileID; Block:BlkNumbers; Buff:PDirBlk);

Abstract:

Reads one block of a file. If block specified is not part of the file then simply zeros the buffer

Parameters:

UserFile is ID of file to read from Block is number of block to read (starting at zero); Buff is buffer to copy data into

Calls: ReadSpiceSegment

Procedure FSSetupSystem(bootchar: integer);

Abstract:

Call this after FSInit to set up the system and print a lot of messages

Parameters:

boot char is ord of key held down to boot

SideEffects: Mounts device from which booted; Mounts all of its partitions Sets AllocDisk's DefaultDeviceName and DefaultPartitionName. Sets FSDirPrefix to be root of current Partition and adds that path to the bottom of the search list

Function FSIsFSDev(name: PathName; var devName: String): integer;

Abstract:

determine whether name is a file that the filesystem knows how to handle

Parameters:

name is a name of a file; devName will be assigned the device name IF NOT FSDevice DevName will be in upper case and does NOT contain the colon.

Returns: 0 if name doesn't contain a : or if dev is one the filesystem knows about; the index of the colon otherwise

```
module FileTypes;

This module exports the types put in the FileType field of File FIBs.  The
types are stored as integers.  PERQ Systems reserves the first 512 types for
their use.  Customers are encouraged to choose numbers > 512 if they invent
new file types

Written by Brad A. Myers  Feb. 2, 1981

Copyright (C) 1980, 1981, 1982  PERQ Systems Corporation
Version Number V1.6
{\\\\\\\\\\\\\\\\\\\\\\\\\} EXPORTS {/////////////////////////}

Const
      UnknownFile = 0;
      SegFile = 1;
      PasFile = 2;
      DirFile = 3;
      ExDirFile = 4;
      FontFile = 5;
      RunFile = 6;
      TextFile = 7;      {for non-Pas text files}
      CursorFile = 8;    {cursor bin files}
      BinaryFile = 9;
      BinFile = 10;      {microcode output}
      MicroFile = 11;
      ComFile = 12;
      RelFile = 13;
      IncludeFile = 14; {included in a pas file}
      SBootFile = 15; {system part of boot file}
      MBootFile = 16; {microcode part}
      SwapFile = 17; {a file used for swapping by compiler or editor; length
                      not set}
      BadFile = 18;  {created by the scavenger}
      ForFile = 19;  { Fortran source file }
      DatFile = 20;  { Fortran unformatted data file }
      PsgFile = 21;  { Fortran pre-seg file }
      ExtFile = 22;  { Fortran external definition file }
      LibFile = 23;  { Fortran library file }
      TempFile = 24;  { Created by Temper }
```

```
module FileUtils;

Filesystem utilities not needed by the system

Written by Brad Myers.  March 5, 1981.

Version Number V1.12
{**********} Exports {**********}

imports FileSystem from FileSystem;

type
  ptrScanRecord = ^ScanRecord;
  ScanRecord    = record
                    InitialCall : boolean;
                    Blk         : DiskAddr;
                    Entry       : Integer;
                    DirName     : PathName;
                  end;

Procedure FSDelete(filename: PathName);
Function  FSScan(scanptr : ptrScanRecord; var name : SimpleName;
                 var id : FileID) : boolean;
Procedure FSRename(SrcName, DestName: PathName);
Function FSMakeDirectory(var DirName: PathName): FileID;
Procedure FSSetSearchList(sList: SearchList);
Procedure FSPopSearchItem(var sList: SearchList);
Procedure FSPushSearchItem(name: PathName; var sList: SearchList);
Procedure FSAddToTitleLine(msg: String); {adds as much of msg as possible to
                                          title line after the current path}
Exception DelError(FileName: PathName);

     Abstract:

          Raised when can't delete file (because not there)

     Parameters:

          FileName is file that can't delete

Exception RenError(msg: String; FileName: PathName);

     Abstract:

          Raised when can't rename file

     Parameters:

          msg is reason can't rename and fileName is file with the problem.
          To print message, use

               "WriteLn('** ',msg,filename);"
```

```
Exception MkDirErr(msg: String; dirName: PathName);
```

Abstract:

Raised when can't make a directory because 1) a file named dirName already exists
2) dirName cannot be entered (bad subdir part)
3) dirName is empty
4) dirName is ROOT.DR (reserved directory name)

Parameters:

msg explains problem with makedir attempt; dirName is name attempted to use. Use

"WriteLn('** ',msg,dirName);"

```
Exception SrchWarn(fileName: PathName);
```

Abstract:

Raised if try to Pop last item or push into last hole of the Search List

Parameters:

'' if Pop; name of item trying to push if Push

Resume: ALLOWED; if resume then does the operation anyway

```
Exception SrchErr(fileName: PathName);
```

Abstract:

Raised if try to Pop empty list or push onto full list for the Search List

Parameters:

'' if Pop; name of item trying to push if Push

Resume: NOT allowed

```
Function FSExtSearch(var SList : SearchList; Extensions: String;
                     var FileName : PathName;
                     var BlksInFile, BitsInLBlk: Integer) :
FileID;
```

```
Exception RenToExist(fileName: PathName);
```

Abstract:

Raised at attempt to rename an existing file. Not raised if renaming a file to its own name (no-op).

Parameters:

fileName - new name that already exists

Resume: ALLOWED; If you wish to rename anyway; just continue and FSRename will delete the DestName; In this case; you should be prepared to accept DelError;

```
Exception RenDir(fileName: PathName);
```

Abstract:

Raised when try to rename a directory.

Parameters:

fileName - name of the source directory.

Resume: ALLOWED; If you wish to rename anyway; just continue and FSRename will do the operation. RenToExist etc. may still be raised.

```
Procedure FSGetFSData(id: FileID; pData: ptrFSDataEntry);
procedure FSSetFSData(id: FileID; pData: ptrFSDataEntry);
procedure FSRemoveDots(var fname: PathName);
```

```
Procedure FSDelete(filename : PathName);
```

Abstract:

Deletes filename from directory and filesystem; fileName is deleted from the current path only (not search lists) if it doesn't contain device or partition info

Parameters:

filename is the name of the file to be deleted

SideEffects: filename is deleted from the current directory if it exists; if not then nothing is done (and the user is not notified)

Calls: DeleteFileID; DestroySpiceSegment

Errors: Raises DelError(fileName) if can't delete file

```
Procedure FSRename(SrcName, DestName: PathName);
```

Abstract:

Changes the name of SrcName to DestName; both are in the current path (not search lists) if not fully specified

Parameters:

SrcName is the name of the file to change and DestName is the name it should be given

Returns:
True if rename is successful; false if can't be done because:

1) - destName already exists
2) - SrcName and destName are in different partitions
3) - SrcName doesn't exist
4) - SrcName or DestName is malformed

SideEffects: The name of the file corresponding to SrcName is changed

Calls: DeleteFileID; DestroySpiceSegment

Errors: Raises RenError(msg, fileName) - if can't rename file where message explains why (do "Write(' ** ',msg,fileName)" in handler) Raises RenToExist(DestName) - if filename already exists; If you wish to rename anyway; just continue and FSRename will delete the DestName; In this case; you should be prepared to accept DelError;

```
Function FSScan(scanptr : ptrScanRecord; var name: SimpleName;
        var id : FileID): boolean;
```

Abstract:

At each call returns the next entry in a directory. The names returned are in random order.

Parameters:

scanPtr is a pointer to a ScanRecord which controls the scan. At the first call, scanPtr^.InitialCall should be set to true and scanPtr^.dirName should be set to the directory to scan through. No fields should be modified by the caller after the initial setting. The dirName field of the scanPtr record is modified to contain the Full path name of the directory. name is set to the name of the file found on this call and id is its fileID; scanPtr is modified after each call so the next call will return the next name in the directory

Returns: True if a valid name and id returned; false if the directory has been exhausted in which case name and id are NOT valid

Function FSMakeDirectory(var dirName: PathName): FileID;

Abstract:

Create a new directory named dirName.

Parameters:

DirName is the name of the directory to create; the name is changed to be the full path name of the directory created

Returns: The fileID of the directory

SideEffects: Creates a file named dirName (appending a ".DR" to end if not there. Sets the FileType field to DirFile; and sets the FileBits to 4096

Errors: Raises MkDirErr(msg, dirName) if 1) a file named dirName already exists 2) dirName cannot be entered (bad subdir part) 3) dirName is empty 4) dirName is ROOT.DR (reserved directory name) where msg describes error. Do not continue from this signal

Procedure FSSetSearchList(sList: SearchList);

Abstract:

Assign the system search list.

Parameters:

sList is new search list. It is a bad idea to not include a partition which contains a full set of system files

SideEffects: Changes system search list

Procedure FSPopSearchItem(var sList: SearchList);

Abstract:

Removes the most recent item from the search list

Parameters:

sList is search list to pop from (it is modified)

Errors: Raises SrchWarn('') if try to pop last item; if continue from it then pops it anyway; Raises SrchErr('') if list empty and try to pop; don't continue from this one

Procedure FSPushSearchItem(name: PathName; var sList: SearchList);

Abstract:

adds name to the front of the search list

Parameters:

name is new name to add to the front of the search list searchList is modified to have name at front

Errors: Raises SrchWarn(name) if try to push into last item; if continue from it then pushes it anyway; Raises SrchErr(name) if list full and try to push; don't continue from this one

Environment: Assumes oldest item in list is at high position (e.g. 5)

Procedure FSAddToTitleLine(msg: String);

Abstract:

adds as much of msg as possible to title line after the current path which is truncated to 35 characters

Parameters:

msg is string to be displayed. The first 43 characters of it are displayed

Side Effects: Changes current window's title line

Procedure FSGetFSData(id: FileID; pData: ptrFSDataEntry);

Abstract:

Returns the FSDataEntry description of a file

Parameters:

id is the FileID for the file that data wanted for pData is a pointer to a data block to which the FSData is copied. Memory for this pointer must be allocated before the call

Procedure FSSetFSData(id: FileID; pData: ptrFSDataEntry);

Abstract:

Changes the FSDataEntry of a file

Parameters:

id is the fileID of the file to be modified pData is the FSDataEntry to set id to. The entire FSDataEntry description of id is changed, so the user should use FSGetFSData to read the FSDataEntry and then change the desired fields only

Side Effects: Changes the FSDataEntry for id

```
Function FSExtSearch(var SList: SearchList; Extensions: String;
      var FileName: PathName; var BlksInFile, BitsInLBlk: Integer): FileID;
```

Abstract:

FSExtSearch performs a breadth-first lookup of a file using a specified searchlist and a list of extensions. The search order is as follows: 1) Try the name with each extension in the current directory. 2) Repeat steps 1 in each path specified in the searchlist. If the file is found, the FileName is changed to be the full file name actually found.

Parameters:

SList - Searchlist to use.

Extensions - List of extensions to try with a single space after each extension. For example, '.Pas .Micro .Cmd .Dfs '. The string must have a single trailing space. A single leading space or a pair of adjacent spaces causes the function to look for the file exactly as typed (no extension appended). Extra spaces are not allowed. If Extensions does not end in a space, then one is added.

FileName - Name of file to find, set to be the full name of the file that was actually found.

BlksInFile - Length of file in blocks.

BitsInLBlk - Bits in last block of file.

Returns: 0 if file not found or id of file

Errors: Raises FSNotFnd if file not found

```
Procedure FSRemoveDots(var fname: PathName);
```

Abstract:

Removes "."s and ".."s from file name leaving full name

Parameters:

fname is file name. It is changed to not have dots.

```
module FTPUtils;

File Transfer Program - Code.

Copyright (C) 1980, 1981, 1982
PERQ Systems Corporation

     Abstract:

          This file contains the code portions for the File Transfer
          Program.

Version Number V6.4
{********************} Exports {********************}

imports Ether10IO from Ether10IO;

Type

   FTPPacket = Record
       Cmd: Char;
       ByteCount: Integer;
       CheckSum: Integer;
       Case Integer Of
           1: (Buffer: Packed Array [0..255] of Char);
           2: (ErrMsg: String);                          { Byte 0 is length
                                                           byte }
           3: (SrcFile: String;
               DestFile: String);
           4: (Name: String);
           5: (Add: EtherAddress);
       End;

   ErrStatus = (OK, TimeOut, ChkSumErr, RawIOErr);

   TransMode = (PERQPERQ, PERQ11, PERQVAX);     { Tells what machines }
                                                { are involved in the
                                                  transfer }

    ByteIntRecord = Packed Record
        Case Integer Of                         { Used to get low order }
            1: (Whole: Integer);                { byte for checksums }
            2: (Lower: 0..255;
                Upper: 0..255);
        End;

    DevTypes = (RS232, FastEther,  EtherNet);        { Valid transfer
                                                       defives }

Function  FTPGetFile(SrcFile,DestFile: String;  IsItText:Boolean;
                     Dev: DevTypes; Mode: TransMode): Boolean;
Function  FTPPutFile(SrcFile,DestFile: String;  IsItText:Boolean;
                     Dev: DevTypes; Mode: TransMode): Boolean;
Procedure FTPChkDev(Dev: DevTypes);
Procedure SendStopVax;
Procedure FTPInit;
```

```
Function  FTPAddRequest(Name: String;  Dev: DevTypes): Boolean;
procedure FTPSetMyAddr(Dev: DevTypes);
procedure FTPQuitNet;

var MyAddr, HisAddr: EtherAddress;

CONST
      MAXALIAS = 10;

VAR NumAlias : 0..MAXALIAS;

Var MyName: Array[1..MAXALIAS] Of String;
    HisName : String;

const FastEType = 1;
      ByteType =  0;

const MaxRecv = 4;

procedure FTPQuitNet;

     Abstract:

          Shut down the net, if needed.

Procedure SendStopVax;

     Abstract:

          This procedure is used to send a StopVax packet.

Function FTPAddRequest(Name: String;  Dev: DevTypes): Boolean;

     Abstract:

          Send a request for an address out on the net.

     Parameters:

          Name is the name that we are to get the address for.

          Dev is the device to use for the transfer.

     Parameters:

          Return true if we could get the address. Return false otherwise.
```

Procedure FTPInit;

Abstract:

This procedure is called to initialize the FTP code.

Side Effects: This procedure will initialize the I/O devices for the machine that it is running on.

Errors: None

Function FTPPutFile(SrcFile,DestFile: String; IsItText:Boolean; Dev: DevTypes; Mode: TransMode): Boolean;

Abstract:

This is the interface routine that will write a file to another machine.

Parameters:

SrcFile is the name of the file, on this machine, that we are to write.

DestFile is the name that is to be used when writing the file on the other machine.

IsItText is a boolean that indicates if the file is a text file. If true then the file is a text file.

Dev is the name of the device that we are to use to transfer the file.

Mode indicates the type of machine that is on the other end of Dev.

Results: Return True if the file was transfered without error. False otherwise.

Side Effects: This procedure will change: CurDevice and CurTMode.

Errors: All errors are indicated by error messages.

Function FTPGetFile(SrcFile,DestFile: String; IsItText:Boolean; Dev: DevTypes; Mode: TransMode): Boolean;

Abstract:

This is the interface routine that will read a file from another machine.

Parameters:

SrcFile is the name of the file that we are to read from the other machine.

DestFile is the name that is to be created on this machine.

IsItText is a boolean that indicates if the file is a text file. If true then the file ic a text file.

Dev is the name of the device that we are to use to transfer the file.

Mode indicates the type of machine that is on the other end of Dev.

Results: Return True if the file was transfered without error. False otherwise.

Side Effects: This procedure will change: CurDevice and CurTMode.

Errors: All errors are indicated by error messages.

procedure FTPSetMyAddr(Dev: DevTypes);

Abstract:

Set the address of this machine and allocate ethernet buffers.

Parameters:

Dev is the current device. It must be Ethernet or FastEther.

Procedure FTPChkDev

Abstract:

This procedure is used to see if the device specivied by Dev need to be serviced. If so then enter the service request routine.

Side Effects: This procedure may change CurChar and CurCharValid.

Errors: None

```
module GetTimeStamp;


GetTimeStamp - Perq get time routine.
J. P. Strait        1 Feb 81.
Copyright (C) PERQ Systems Corporation, 1981.

     Abstract:

          GetTimeStamp implements the read-time-as-TimeStamp function for
          the Clock module.  See the Clock module for more details.

          Design: GetTimeStamp is a separate module so that it may be
          imported into the resident system without importing all the other
          Clock routines.  Once virtual memory is implemented, GetTimeStamp
          and Clock should be merged into a single module.

Version Number V1.4
{//////////////////////////} Exports {\\\\\\\\\\\\\\\\\\\\\\\\}

const GetTSVersion = '1.4';

type TimeStamp = packed record
       { the fields in this record are ordered this way to optimize bits }
       Hour:   0..23;
       Day:    1..31;
       Second: 0..59;
       Minute: 0..59;
       Month:  1..12;
       Year:   0..63;  { year since 1980 }
       end;

     TimeReference = record
                       Lower: Integer;
                       Upper: Integer
                       end;

procedure GetTStamp( var Stamp: TimeStamp );

var PastStamp: TimeStamp;
    Past: TimeReference;

procedure GetTStamp( var Stamp: TimeStamp );

     Abstract:

          returns a timeStamp for the current time

     Parameters:

          Stamp is set to be the stamp for the current time
```

```
module gpib;

    Abstract:

        Support routines for PERQ GPIB devices.  The package maintains a
        buffer (gpCommandBuffer) which holds either data bytes (sent with
        procedure gpPutByte) or Auxiliary commands (sent with
        gpAuxCommand). The buffer is sent to the 9914 when full, or when
        gpFlushBuffer is called.  If the buffdr has data bytes when
        gpAuxCommand is called, it will do a gpFlushBuffer.  Similarly,
        when gpPutByte is called, it will flush the buffer, if auxiliary
        commands are in gpCommandBuffer.

        written by Brian Rosen
        Copyright(C) 1980, PERQ Systems Corporation

Version Number V1.5


exports

 const
       GpibVersion = '1.5';
       gpBufSize = 12;
       gpBufMax = 11; {gpBufSize - 1}

 { the following codes are the IEE488-1975 Controller Command Codes
   they are issued by the Controller-In-Charge while asserting ATN }
       gpacg = #000; {addressed group command}
       gpdcl = #024; {device clear}
       gpget = #010; {group execute trigger}
       gpgtl = #001; {go to local}
       gplag = #040; {listen address group}
       gpllo = #021; {local lockout}
       gpmla = #040; {my listen address}
       gpmta = #100; {my talk address}
       gpmsa = #140; {my secondary address}
       gpppc = #005; {parallel poll configure}
       gpppe = #140; {parallel poll enable}
       gpppd = #160; {parallel poll disable}
       gpppu = #025; {parallel poll unconfigure}
       gpscg = #140; {secondary copmmand group}
       gpsdc = #004; {selected device clear}
       gpspd = #061; {serial poll disable}
       gpspe = #060; {serial poll enable}
       gptct = #011; {take control}
       gptag = #100; {tahk address group}
       gpuag = #020; {universal address group}
       gpunl = #077; {unlisten}
       gpunt = #137; {untalk}

 type  { these commands are the major state change control commands
         of the TMS9914 chip which forms the interface to the GPIB
         Consult the TI documentation on the TMS9914 for more information}

       {These definitions are order dependent}
```

```
        gpAuxiliaryCommands = (gpswrst,   {Chip Reset}
                               gpdacr,    {Release DAC holdoff}
                               gprhdf,    {Release RFD holdoff}
                               gphdfa,    {Holdoff all data}
                               gphdfe,    {Holdoff on End}
                               gpnbaf,    {Set NewByteAVailable false}
                               gpfget,    {Force Group Execute Trigger}
                               gprtl,     {Return to Local}
                               gpfeoi,    {force End or Identify}
                               gplon,     {Listen Only}
                               gpton,     {Talk Only}
                               gpgts,     {GoTo Standby}
                               gptca,     {Take Control Asynchronously}
                               gptcs,     {Take Control Synchronously}
                               gprpp,     {Request Parallel Poll}
                               gpsic,     {Set Interface Clear}
                               gpsre,     {Set Remote Enable}
                               gprqc,     {Request Control}
                               gprlc,     {Release Control}
                               gpdai,     {Disable All Interrupts}
                               gppts,     {Pass Through next Secondary}
                               gpstdl,    {Set T1 Delay}
                               gpshdw);   {Shadow Handshake}
        gpParmType = (gpOff, gpOn, gpDontCare);  {parameters for Aux
                                                  Commands}
        gpByte = 0..255;    {Data byte for gpib transactions}
        gpRange = 0..gpBufMax;
        gpDeviceAddress = 0..31;          {legal addresses for devices on
                                           GPIB}
        gpBuffer = packed array [gpRange] of gpByte;
        gppBuffer = ^gpBuffer;

 var    gpCommandBuffer: gppBuffer;       {place to put commands}
        gpBufPtr: 0..gpBufMax;           {pointer to gpCommandBuffer}
        gpHaveDataBytes, gpHaveAuxiliaryCommands: boolean;{true if buffer
                                                           in use}
{   The package maintains a buffer (gpCommandBuffer) which holds
    either Data bytes (sent with proceedure gpPutByte)
    or Auxialiary Commands (sent with gpAuxCommand)
    The buffer is sent to the 9914 when full, or when ghForceBuffer
    is called.  If the buffer has data bytes when gpAuxilairyCommand is
    called, it will do a gpForceBuffer.  Similarly, when gpPutByte
    is called, it will force the buffer if auxiliary commands are in
    gpCommandBuffer                                                    }

        { Initialze GPIB package, called once only, turns off tablet }
procedure gpInit;

        { Send an auxiliary command to TMS9914
          some commands require a parameter (gpOff/gpOn) }

procedure gpAuxCommand(gpCmd: gpAuxiliaryCommands; gpParm:gpParmType);

        { Put a data byte or a Control byte out on the data bus
          TMS9914 must be in Controller Actives State if the byte is a
          controller command byte.  Must be in Talk Only if a data byte }
```

```
procedure gpPutByte(gpData: integer);

          { Sends all bytes in buffer }

procedure gpFlushBuffer;

          { Set TMS9914 to be a Talker, set a device to be a listener
            This procedure takes control of the bus, unlistens and untalks
            all devices (including itself), and sets a listener with
            MyListenAddress then sets TMS9914 to be the talker with
            TalkONly }

procedure gpITalkHeListens(gpAddr: gpDeviceAddress);

          { Set TMS9914 to be a Listener, set a device to be a talker
            This procedure takes control of the bus, unlistens and untalks
            all devices (including itself), and sets a talker with
            MyTalkAddress then sets TMS9914 to be the listener with
            ListenONly}

          { turn the BitPad back on again }

procedure gpTbltOn;

          { turn the BitPad (device addres #10) off }

procedure gpTbltOff;

          { Send a buffer of user data to the 9914 }

procedure gpSend(var gpBuf: gppBuffer; gpCount: gpRange);

          {Get a buffer of data from the 9914 (Not implemented yet) }

procedure gpReceive( var gpBuf: gppBuffer; gpCount: gpRange);

          { Get a byte of data from the GPIB }

function  gpGetByte: gpByte;

procedure gpCleanup;

          { cleans up after the GPIB package, turns the tablet backon }

exception GPIBerror(SoftStatus: integer);
```

Abstract:

Raised when GPIB encounters an error indication in softstatus from UnitIO or IOCRead. The condition should be corrected and the operation retried. The most likely error is a timeout: IOETIM (See IOErrors).

```
procedure gpFlushBuffer;

     var
     address: double;

     begin

     if gpHaveAuxiliaryCommands
     then GPB_UnitIO( Recast(gpCommandBuffer,IOBufPtr), IOWriteRegs
     , gpBufPtr,  gpStatPtr)

     else if gpHaveDataBytes
     then GPB_UnitIO( Recast(gpCommandBuffer,IOBufPtr), IOWrite
     , gpBufPtr, gpStatPtr)

     else gpStatPtr^.SoftStatus := IOEIOC;

     gpHaveAuxiliaryCommands := false;
     gpHaveDataBytes := false;
     gpBufPtr := 0;

     if gpStatPtr^.SoftStatus <> IOEIOC
     then raise GPIBerror(gpStatPtr^.SoftStatus);

     end;

procedure gpITalkHeListens(gpAddr: gpDeviceAddress);

     begin

     gpAuxCommand(gptca,gpDontCare);  {take over bus}
     gpAuxCommand(gpton,gpOff);       {I am not a talker}
     gpAuxCommand(gplon,gpOff);       {I am not a listener}

     gpPutByte(gpunl);                {unlisten all devices}
     gpPutByte(gpunt);                {untalk all devices}
     gpPutByte(gpmla+gpAddr);         {set MyListenAddress}

     gpAuxCommand(gpton,gpOn);        {I will become a Talker}
     gpAuxCommand(gpgts,gpOn);        {Go to it}
     gpFlushBuffer;

     end;

procedure gpCleanup;

     begin

     if  (TypePointDev = TheGPIBBitPad)
     and (gpTabMode <> OffTablet)
     then IOSetModeTablet( gpTabMode )
```

```
module Helper;

WJHansen      Jan 82.
Copyright (C) PERQ Systems Corporation, 1982.

    Abstract:

        Reads an index file and presents options for assistance to the
        user.

Version Number V1.4


exports

imports FileDefs from FileDefs;        {for PathName}

procedure GiveHelp(FName:PathName);

procedure GiveHelp(FName:PathName);

    Abstract:

        Reads a help index and displays it.  Lets user ask for information
        on topics in the index and displays the files containing those
        topics.

    Parameters:
        FName - Name of the file containing the index.  The path to this
            file is used as the path to the individual help files.
```

```
module IOClock;

IOClock - 'Private' Clock type and variable declarations - available
to the IO subsystem.  Clock routines.

Copyright (C) 1982, PERQ Systems Corporation

     Abstract:

          IOClock exports variables, constants, and procedures the IO
          subsystem uses to do Clock IO.

Version Number V0.2
{*****************************}  Exports  {*****************************}

imports IO_Unit from IO_Unit;

procedure Clk_Initialize;
procedure Clk_UnitIO( Bufr    : IOBufPtr;
                      Command : IOCommands;
                      ByteCnt : integer;
                      StsPtr  : IOStatPtr );
procedure Clk_Interrupt;

procedure Clk_Interrupt;

procedure Clk_Initialize;

procedure Clk_UnitIO( Bufr    : IOBufPtr; Command : IOCommands; ByteCnt :
     integer; StsPtr  : IOStatPtr );
```

```
module IODisk;

{------------------------------------------------------------------------
{
{ IODisk - Contains Disk IOUnit Function for EIO and CIO Disks. TV.
{          'Private' HardDisk type and variable declarations - available
{          to the IO subsystem.  Disk routines. AGR.
{
{ Copyright (C) 1982, 1983 PERQ Systems Corporation
{
{ Abstract:
{       IODisk exports variables, constants, and procedures the IO
{       subsystem uses to do disk IO.
{
{------------------------------------------------------------------------}

{ $Version V0.6 for POS}
{------------------------------------------------------------------------

{**************************}  Exports  {**************************}

Imports IO_Unit From IO_Unit;


Procedure Dsk_Interrupt;
Procedure Dsk_Initialize;
Procedure Dsk_UnitIO( Unit    : UnitRng;
                      Bufr    : IOBufPtr;
                      Command : IOCommands;
                      ByteCnt : Integer;
                      DskAdr  : Double;
                      HdPtr   : IOHeadPtr;
                      StsPtr  : IOStatPtr );
```

```
module IOErrMessages;

    Abstract:

        This module exports a procedure to return an  error string for a
        disk error

        Written by : Brad A. Myers  May 12, 1981
        Copyright (C) 1981 - PERQ Systems Corporation
Version Number V1.5

{//////////////////////////} EXPORTS {\\\\\\\\\\\\\\\\\\\\\\\\}

Function IOErrString(err: integer): String;

Function IOErrString(err: integer): String;

    Abstract:

        Returns a string describing the error number

    Parameters:

        err is the error number returned by UnitIO

    Returns: A string describing the error
```

```
module IOErrors;

    Abstract:

        I/O System Error Code Definitions

        Copyright (C) 1981,1982,1983 - The PERQ Systems Corporation
Version Number V1.8

exports

Imports SystemDefs from SystemDefs;  {using Ether3MBaud}

Const
        IOEIOC = 1;         { IO Complete }
        IOEIOB = 0;         { IO Busy }
        IOEBUN = -1;        { Bad Unit Number }
        IOENBD = -2;        { Raw Block IO to this device is not implemented }
        IOEWRF = -3;        { Write Failure }
        IOEBSE = -4;        { BlockSize Error }
        IOEILC = -5;        { Illegal Command for this device }
        IOENHP = -6;        { Nil Header Pointer }
        IOEADR = -7;        { Address Error }
        IOEPHC = -8;        { Physical Header CRC Error }
        IOELHC = -9;        { Logical Header CRC Error }
        IOEDAC = -10;       { Data CRC Error }
        IOEDNI = -11;       { Device Not Idle }
        IOEUDE = -12;       { Undefined Error! }
        IOENCD = -13;       { Device is not a character device }
        IOECBF = -14;       { Circular Buffer Full }
        IOELHS = -15;       { Logical Header SerialNum Mismatch }
        IOELHB = -16;       { Logical Header Logical Block Number Mismatch }
        IOECOR = -17;       { Cylinder Out of Range }
        IOEDNR = -18;       { Device not ready }
        IOEMDA = -19;       { Missing data address mark }
        IOEMHA = -20;       { Missing header address mark }
        IOEDNW = -21;       { Device not writable }
        IOECMM = -22;       { Cylinder mis-match }
        IOESNF = -23;       { Sector not found }
        IOEOVR = -24;       { Overrun }
        IOEUEF = -25;       { Undetermined equipment fault }
        IOESOR = -26;       { Sector out of range }
        IOETIM = -27;       { Time out error }
        IOEFRS = -28;       { Floppy recalibrate done }
        IOEDRS = -29;       { Disk recalibrate done }
        IOET0  = -30;       { Can't find track zero }
        IOECDI = -31;       { Data supplied to configuration command is bad }
        IOERDI = -32;       { Register data for WriteRegs command is bad }
        IOEBAE = -33;       { Buffer alignment error }
        IOENOC = -34;       { Not on Cylinder }
        IOEABN = -35;       { Abnormal Error }
        IOELHE = -36;       { Logical Header Mismatch }
        IOESME = -37;       { State Machine Error }
        IOESKE = -38;       { Drive Seek Error }
```

```
        IOEFLT = -39;       { Drive Fault }
        IOEDNS = -40;       { Device not supported }
        IOEPHM = -41;       { Physical Header Mismatch }

{$ifc Ether3MBaud then}
        IOEPTL = -42;       { Ether3 - received packet too large }
{$endc}

        IOEEND = -43;       { End of data }
        IOEFRA = -44;       { Framing error }
        IOEPAR = -45;       { Parity error }

    IOEFirstError = -45;

    IOELastError = 0;
```

```
module IOFloppy;

IOFloppy - Floppy IO routines.
Copyright (C) 1982, 1983 PERQ Systems Corporation

    Abstract:

        IO_Floppy exports procedures to perform IO on the floppy.

        Design: 1) UnitIO must increment and decrement the IOCount of the
        segments which are involved in IO. 2) Segment faults must *never*
        happen while interrupts are off.

Version Number V0.6
{*****************************}  Exports
{*****************************}
Imports IO_Private from IO_Private;

Procedure FLP_Initialize;              { Floppy Initialization     }
Procedure FLP_Interrupt;               { Floppy interrupt handler  }
Procedure FLP_UnitIO(                  { Floppy UnitIO routine     }
              Bufr: IOBufPtr;
              Command: IOCommands;
              ByteCnt: Integer;
              LogAdr: Double;
              StsPtr: IOStatPtr);

Procedure FLP_PutStatus(               { Set status on device Unit }
              var StatBlk: DevStatusBlock);
Procedure FLP_GetStatus(               { Read status on device Unit }
              var StatBlk: DevStatusBlock);

Procedure FLP_Initialize;              { Floppy Initialization     }

Procedure FLP_Interrupt;

    Abstract:

        FloppyIntr handles a floppy interrupt. If the handler is waiting
        for an interrupt The interrupt cause is read and cleared. The
        Interrupt and Attention causes are held for the low level handler

        If the interrupt is an attention then the current cylinder is set
        to -1 to force a seek on the next operation.
```

```
module IOGPIB;

IOGPIB - 'Private' GPIB type and variable declarations - available
to the IO subsystem.  GPIB routines.

Copyright (C) 1982, 1983, PERQ Systems Corporation

     Abstract:

          IOGPIB exports variables, constants, and procedures the IO
          subsystem uses to do GPIB IO.

Version Number V0.7
{*****************************}  Exports  {*****************************}

imports IO_Unit from IO_Unit;

const  { Fudge factors for BitPad }
  GPIBxFudge =   38;  { actual range in X and Y for BitPad: 0..2200 }
  GPIByFudge = 1061;  { of TabABsX  : 0..1100 }
                      { of TabAbsY  : 0..1100 }
                      { of TabRelX  : -38..1062  limited to 0..767 }
                      { of TabRelY  : 1061..-39  limited to 1023..0 }

{ Update tells us if the puck was lifted off the pad.  Our interrupt routine
  clears it everytime it gets data from the gpib.  The tablet update routine
  adds one to it everytime it updates the tablet values.  If the tablet
  update routine finds that we haven't set it to zero after a couple of
  times, it assumes the puck is off the bad.  Tablet update is in IOVideo. }

var
  GPIBTabBuf : packed record
    X : integer;
    Y : integer;
    Buttons : integer;
    Update : integer
    end;
  GPIBTabletState : integer;
  GPIBIntMask : integer;     { the interrupt mask for the gpib }

procedure GPB_Interrupt;
procedure GPB_Initialize;
function  GPB_ReadChar( var Ch : char ): integer;
function  GPB_WriteChar( var Ch : char ): integer;
procedure GPB_UnitIO( Bufr    : IOBufPtr;
                      Command : IOCommands;
                      ByteCnt : integer;
                      LogAdr  : Double;
                      StsPtr  : IOStatPtr );
procedure GPB_GetStatus( var StatBlk : DevStatusBlock );
```

```
procedure GPB_Interrupt;

    Abstract:

        GPB_Interrupt handles an interrupt from the GPIB.

procedure GPB_Initialize;

    Abstract:

        Initialize the GPIB.

function GPB_ReadChar( var Ch : char ): integer;

    Abstract:

        Do character reads from the GPIB

function GPB_WriteChar( var Ch : char ): integer;

    Abstract:

        Do one character writes to the GPIB.  (Use GPB_UnitIO)

procedure GPB_UnitIO( Bufr    : IOBufPtr; Command : IOCommands; ByteCnt :
    integer; LogAdr  : Double; StsPtr  : IOStatPtr );

    Abstract:

        Do IO to the GPIB

        Notes The LogAdr is a count of the number of jiffies to wait
        before timing out an operation and resetting the GPIB.  It is
        recast into a long.

procedure GPB_GetStatus( var StatBlk : DevStatusBlock );

    Abstract:

        Provide status information about the GPIB.  This is here only to
        provide compatibility with the old system.  The recommended way to
        get status information is to use UnitIO and the IOSense command.
```

```
module IOKeyboard;

IOKeyboard - Keyboard IO routines.

Copyright (C) 1982, 1983 PERQ Systems Corporation

     Abstract:

          IOKeyboard exports procedures to perform IO on the keyboard.

Version Number V0.4
{******************************}  Exports
{******************************}

Imports IO_Private from IO_Private;

Const
        CtrlC = chr(#3);
        CtrlS = chr(#23);
        CtrlQ = chr(#21);
        BlamCh = Chr(#303);       { untranslated shift-control-C }
        DumpCh = Chr(#304);       { untranslated shift-control-D }

var
  KTBuf : CirBufPtr;  { Keyboard translated buffer }

procedure Key_Initialize;

Function Key_ReadChar( Unit : UnitRng; var Ch: char): integer;

     { disable/enable keyboard interrupts }

Procedure Key_Disable( var OldKeyEnable: Boolean );
Procedure Key_Enable( OldKeyEnable: Boolean );

Procedure Key_Clear;          { clear the IO type-ahead buffer }

Procedure Key_Interrupt;

Function Key_TLate(Ch : Char): Char;
```

procedure Key_Initialize;

Abstract:

Keyboard initialization logic.

Function Key_TLate(Ch : Char): Char;

Abstract:

Translate a raw key board character.

Parameters:
Ch- The character to be translated.

Returns: A valid ascii (less than #200) character.

Function Key_ReadChar( Unit : UnitRng; var Ch: char): integer;

Abstract:

Reads a character from keyboard and returns a completion or error code.

Parameters:
Ch - character to read.

Returns:

A condition code as defined in the module IOErrors.

Procedure Key_Interrupt;

Abstract:

Key_Interrupt processes KeyBoard interrupts by copying characters from the KeyBoard buffer into the (misnamed) translated keyboard buffer (KTBuf). Control-C, Control-Shift-C, Control-Shift-D, Control-S, HELP and Control-Q are processed also.

```
procedure Key_Disable( var OldKeyEnable: Boolean );
```

Abstract:

Key_Disable is used to disable keyboard interrupts. This is used to delay processing of control-c, control-shift-c and control-shift-d at critical times. The old value of the keyboard interrupt enable is returned and must be passed back to Key_Enable when re-enabling keyboard interrupts. Characters typed while keyboard interrupts are disabled are remembered. When keyboard interrupts are re-enabled, the characters are processed.

Parameters:
OldKeyEnable - set to the old value of the enable.

```
procedure Key_Enable( OldKeyEnable: Boolean );
```

Abstract:

Key_Enable is used to enable keyboard interrupts. The old value of the keyboard interrupt enable (as returned from Key_Disable) must be passed to Key_Enable when re-enabling keyboard interrupts. If characters were typed while keyboard interrupts were enabled, Key_Enable calls Key_Interrupt to process those characters. The master interrupt control (INTON and INTOFF QCodes) must be on when this procedure is called.

Parameters:
OldKeyEnable - the old value of the enable.

```
procedure Key_Clear;
```

Abstract:

Key_Clear clears the keyboard type-ahead buffer.

```
module IOPointDev;

IOPointDev - 'Private' PointDev type and variable declarations;
available to the IO subsystem.  PointDev routines.

Copyright (C) 1982, 1983 PERQ Systems Corporation

     Abstract:

          IOPointDev exports variables, constants, and procedures the IO
          subsystem uses to do PointDev IO.

Version Number V0.4
{**************************}  Exports  {**************************}

imports IO_Unit from IO_Unit;

const  { Fudge factors for Kriz Tablet }
  KrizXFudge =   64;  { actual range of X is 0..895, of Y is 0..1151 }
  KrizYfudge = 1087;  { of TabAbsX : 0..895,  of TabAbsY : 0..1151 }
                      { of TabRelX : -64..831    limited to 0..767 }
                      { of TabRelY : 1087..-64   limited to 1023..0 }
type
  pPointBuf = ^PointBuf;
  PointBuf = packed record
    XPos:    integer;          { X position of the pointer }
    YPos:    integer;          { Y position of the pointer }
    Buttons: integer;          { indicates which buttons are pressed }
    Filler:  integer;          { so micro code's area is quad word
                                 aligned }
    UCodeArea : packed array[0..3] of integer
    end;

{ On the buttons, the least significant bit indicates that the rightmost
  button is pressed.  The next bit indicates that the middle button is
  pressed.  The third bit indicates that the leftmost button is pressed.
  Bit seven indicates that the puck is off the pad. }

var
  KrizInfo : pPointBuf;

procedure Ptr_Initialize;
procedure Ptr_PutStatus( var StatBlk : DevStatusBlock );
procedure Ptr_GetStatus( var StatBlk : DevStatusBlock );
procedure Ptr_UnitIO( Bufr    : IOBufPtr;
                      Command : IOCommands;
                      ByteCnt : integer;
                      StsPtr  : IOStatPtr );
procedure Ptr_Interrupt;
procedure Ptr_Interrupt;

     Abstract:

          Handles PointDev interrupts, these will be the only responses for
          commands issued to the tablet.
```

```
procedure Ptr_Initialize;

     Abstract:

          Prepare for IO to the PointDev.  Determine if tablet is connected
          by enabling it for a short while for it to send current
          coordinates.  If no data is received within a short time, flag the
          tablet as not being connected.

procedure Ptr_PutStatus( var StatBlk : DevStatusBlock );

     Abstract:

          Put a staztus to the PointDev, otherwise known as the KrizTablet.
          The tablet can only be enabled or disabled.

     Parameters:

          StatBlk - address of status block

procedure Ptr_GetStatus( var StatBlk : DevStatusBlock );

     Abstract:

          Get status from the PointDev.

     Parameters:

          StatBlk - address of status block

procedure Ptr_UnitIO( Bufr : IOBufPtr;
     Command : IOCommands; ByteCnt : integer;
     StsPtr  : IOStatPtr );
     Abstract:

          Execute IOSense and IOConfig commands for IOPointDev.

     Parameters:

     Bufr - buffer for data transfers, if requested
     Command - operation to be performed on the device
     ByteCnt - number of bytes to be transferred
     StsPtr - resultant status from the operation
```

```
module IORS;

IORS - RS IO routines.

     Abstract:

          IORS exports procedures to perform IO on RS232 ports. These
          routines are RS232 specific interrupt, initialization, and general
          IO routines, and are exported to the IO subsystem modules. (
          IO_Unit, IO_Init )

          Notes: Speech is on an RS232 line, so all speech IO goes through
          this module.

          Copyright (C) 1982, 1983 PERQ Systems Corporation
Version Number V0.9

{*****************************}  Exports
{*****************************}

Imports IO_Private from IO_Private;

var
  RSIntMask : integer;       { mask of interrupts to enable for RS232 }

procedure Rs_Initialize;

Function  Rs_ReadChar (Unit : UnitRng ; var Ch: char ): integer;
Function  Rs_WriteChar(Unit : UnitRng ; Ch: char ): integer;

Procedure Rs_PutStatus(Unit : UnitRng ; var UserStatus : DevStatusBlock);

Procedure RsA_Interrupt;  { Interrupt handler for RS232 port 'A' }
Procedure RsB_Interrupt;  { Interrupt handler for RS232 port 'B' }
Procedure Spc_Interrupt;  { Interrupt handler for Speech }

procedure RS_UnitIO (  Unit: UnitRng;
                       Bufr: IOBufPtr;
                       Command: IOCommands;
                       ByteCnt: integer;
                       StsPtr: IOStatPtr );

procedure RS_Initialize;

     Abstract:

          RS232 initialization logic.
```

```
Function RS_ReadChar(Unit : UnitRng ; var Ch: char ): integer;
```

Abstract:

Reads a character from RS232 port A and returns a completion or error code.

Parameters:

Ch - character read.

Returns: A condition code as defined in module IOErrors.

```
Function RS_WriteChar(Unit : UnitRng; Ch: char): integer;
```

Abstract:

Writes a character to RS232 port A or B and returns a completion or error code.

Parameters:
Unit - Port (A or B) to read from.

Ch - character to write.

Returns: A condition code as defined in module IOErrors.

```
Procedure RS_PutStatus (Unit: UnitRng; var UserStatus:DevStatusBlock);
```

Abstract:

Sets port's characteristics. Translates old put status command into a configure command and a writeregs command, mapping the old status block into a set of Serial IO controller registers and the baud rate for the configure command.

Parameters:
Unit - device whose characteristics are to be set.
UserStatus - device status block containing characteristics to be set.

SideAffects:

Sets SIO register settings not mapped from the old device status block to their defaults.

```
Procedure RS_UnitIO (  Unit: UnitRng; Bufr: IOBufPtr; Command: IOCommands;
     ByteCnt: integer; StsPtr: IOStatPtr );

     Abstract:

          Unit IO operations to RS232 ports.

     Parameters:
          Unit - the device.

          Bufr - buffer for data transfers, if requested.

          Command - operation to be performed on the port.

          ByteCnt - number of bytes to be transferred.

          StsPtr - resultant status from the operation.

Procedure RSA_Interrupt;

     Abstract:

          This is the interrupt routine for the RS232 port 'A'.

Procedure Spc_Interrupt;

     Abstract:

          This is the interrupt routine for the Speech port.

Procedure RSB_Interrupt;

     Abstract:

          This is the interrupt routine for the RS232 port 'B'.
```

```
module IOVideo;

    Abstract:

        Private Video type and variable declarations.
        IOVideo exports variables, constants, and procedures that the IO
        subsystem uses to do video manipulation

        Copyright (c) 1982, 1983, PERQ Systems Corporation

Version Number V0.4

exports

imports IO_others from IO_others;
imports IO_Private from IO_Private;

const
  TabIgnore = 2;        { number of points to ignore after when ignoring }

var
  Cursor: CurPatPtr;                { Cursor Pattern }

  CursorX, CursorY: integer;        { new cursor coordinates }
  OldCurY,                          { previous Cursor Y position }
  OldCurX: integer;                 { previous Cursor X position MOD 8 }

  PointX, PointY: integer;          { the point of the cursor }

  TabCount : integer;               { number of points left to ignore }

  CursF: integer;                   { function currently in use}
  BotCursF: integer;                { function for area below used area}
  BotComplemented: boolean;         { whether bot is complemented or not}
  TabMode: TabletMode;              { Current mode of the tablet }
  CCursMode: CursMode;              { Current mode of cursor }
  newFunct: Boolean;                { Tells when have a new function to
                                          insure that cursor redisplayed }

Procedure Vid_Initialize;           { Initialization for the video }
Procedure Vid_Interrupt;            { Interrupt Routine for the video device }
Function  Vid_SetUpUDevTab: pointer;  { Set up the pointers in the micro-  }
                                    { code device table that the micro needs }

procedure Vid_Initialize;

    Abstract:

        Initialize all the variables in IO_Others that we set, set up the
        default cursor, enable Video interrupts.

        WARNING: This must be called AFTER ScreenInit.
```

Procedure Vid_Interrupt;

Abstract:

Vid_Interrupt (formerly TabIntr) handles the screen retrace interrupt. It smoothes the tablet data and updates the displayed cursor position (if it is visible and has moved).

function Vid_SetUpUDevTab: pointer;

Abstract:

Set up the screen control block for the microcode. Does not use Screen package.

Returns: Pointer to the screen buffer.

```
module IOZ80;


IOZ80 - 'Private' Z80 type and variable declarations - available
to the IO subsystem.  Z80 routines.

Copyright (C) 1982, PERQ Systems Corporation

     Abstract:

          IOZ80 exports variables, constants, and procedures the IO
          subsystem uses to do Z80 IO.

Version Number V0.4
{**************************}   Exports   {**************************}

imports IO_Unit from IO_Unit;

procedure Z80_Initialize;
procedure Z80_UnitIO( Bufr    : IOBufPtr;
                      Command : IOCommands;
                      ByteCnt : integer;
                      LogAdr  : double;
                      StsPtr  : IOStatPtr );
procedure Z80_Interrupt;

procedure Z80_Interrupt;

     Abstract:


          Handle interrupts from the clock.  Interrupts from the clock are
          responses to commands sent to the clock.

procedure Z80_Initialize;

     Abstract:


          Initialize the clock by enabling interrupts.  This routin can be
          called any number of times.  (Thus the check for nill pointers in
          the device table.) An initial sense is done to determine if the
          clock is supported by the hardware.

procedure Z80_UnitIO( Bufr    : IOBufPtr;
     Command : IOCommands;
     ByteCnt : integer;
     LogAdr  : double;
     StsPtr  : IOStatPtr );

     Abstract:


          Do IO to the Z80 device.  High volume read and write allow loading
          and reading Z80 memory.  Sense to determine Z80 version number,
          Writeregs to call a location in Z80 memory.
```

Parameters:
Bufr - buffer for data transfers, if requested.
Command - operation to be performed on the device.
ByteCnt - number of bytes to be transferred.
LogAdr - address used for WriteRegs.
StsPtr - resultant status from the operation.

```
module IO_Init;

IO_Init - Initialize the IO system.

Copyright (C) 1982, 1983, PERQ Systems Corporation

    Abstract:

        IO_Init initializes the Interrupt Vector Table, the Device Table
        and associated buffers, the Screen Package, the tablet and cursor,
        and the Z80.  IO_Init imports various device dependent modules,
        which contain device dependent initialization code.

        This module is based on Version V5.9 of old I/O board's IO_Init
        support module, written by Miles Barel and modified by numerous
        times by just about every engineer at PERQ Systems Corporation.

Version Number V7.12
{*****************************}  Exports
{*****************************}

Procedure InitIO;
Procedure ReInitDevices;



Procedure InitIO;

    Abstract:

        InitIO initializes the Interrupt Vector Table, the Device Table
        and associated buffers, the Screen Package, the tablet and cursor,
        and the Z80.

procedure ReInitDevices;

    Abstract

        Initialize Z80 devices which were not initialized before due to
        lack of Z80 support.  (The Z80 proms aren't large enough to
        support all the devices.  Thus, the system must load the Z80
        program into the Z80 ram sometime during initialization.  After it
        does this, it calls us so that all the devices work.)
```

```
module IO_Others;

IO_Others - Miscellaneous IO routines.
Miles A. Barel  ca. 1 Jan 80.
Copyright (C) 1980, 1982, 1983 PERQ Systems Corporation

     Abstract:

          IO_Others exports routines for the Cursor, Table, Screen, Time,
          and Keyboard.

Version Number V6.8

{*****************************}  Exports
{*****************************}

Imports SystemDefs from SystemDefs;

{ tablet/cursor procedures }

Type
  CursFunction = (CTWhite,  CTCursorOnly, CTBlackHole, CTInvBlackHole,
                  CTNormal, CTInvert,     CTCursCompl, CTInvCursCompl);
  TabletMode = (relTablet, scrAbsTablet, tabAbsTablet, offTablet);
  CursMode = (OffCursor, TrackCursor, IndepCursor);
  CursorPattern = array[0..63,0..3] of integer;
  CurPatPtr = ^CursorPattern;
  TabletType = (NoPointDev, KrizTablet, GPIBBitPad );

Var
  TabRelX, TabRelY : integer;  { tablet relative coordinates }
  TabAbsX, TabAbsY : integer;  { tablet absolute coordinates }
  TabFinger  : boolean;       { finger on tablet }
  TabSwitch  : boolean;       { switch pushed down }
  TabWhite   : boolean;       { True if white or left button down }
  TabGreen   : boolean;       { True if green or right button down }
  TabBlue    : boolean;       { True if blue button down }
  TabYellow  : boolean;       { True if yellow or middle button down }
  TabMouse   : integer;       { Actual output from mouse }
  DefaultCursor: CurPatPtr;   { default cursor pattern }
  RealRelTablet : boolean;    { indicate if table in true relative mode }
  TypePointDev : TabletType;  { tells which tablet is the pointer }
  BitPadTimeOut : integer;    { how long before puck off bit pad }
  KrizTabConnected : boolean; { true if KrizTablet connected  }
  GPIBpadConnected : boolean; { ture iff the BitPad is connected  }
  TabLeft    : boolean;       { true if left or white button down }
  TabMiddle : boolean;        { true if middle or yellow button down }
  TabRight  : boolean;        { true if right or green button down }

Procedure IOLoadCursor(Pat: CurPatPtr; pX, pY: integer);
                                        { load user cursor pattern }
Procedure IOReadTablet(var tabX, tabY: integer); { read tablet coordin
Procedure IOSetFunction(f: CursFunction);
Procedure IOSetModeTablet(m: TabletMode); { set the mode to tell what kind
                                            of tablet is currently in use }
Procedure IOCursorMode (m: CursMode); { if track is true, then Tablet
```

```
                                        coordinates are copied every 1/60th
                                        second into the cursor position. if
                                        indep, then coordinates are changed
                                        only by user.  If off, then no
                                        cursor displayed }
Procedure IOSetCursorPos(x,y: Integer); { if trackCursor is false, then sets
                                         cursor x and y pos.  If tracking,
                                         then sets both tablet and cursor. }
Procedure IOSetTabPos (x,y: Integer);  { if trackCursor is false, then sets
                                         tablet x and y pos.  If tracking,
                                         then sets both tablet and cursor }
Procedure IOReadCursPicture(pat: CurPatPtr; var px, py: integer);
                                       { copies current cursor picture into
                                         pat and sets px and py with the
                                         offsets for the current cursor }
Procedure IOGetTime(var t: double);  { Get the double word 60 Hertz time }

            {Procedure to change screen size}

Procedure IOScreenSize(newSize: integer; Complement: Boolean);
                                       { newSize is number of scan lines in
                                         new screen; must be a multiple of
                                         128. Complement tells whether the
                                         rest of the screen should be the
                                         opposite color from the displayed
                                         part }

            { disable/enable keyboard interrupts }

Procedure IOKeyDisable( var OldKeyEnable: Boolean );
Procedure IOKeyEnable( OldKeyEnable: Boolean );  { enable keyboard i
                                                   nterrupts }
Procedure IOKeyClear;     { clear the IO type-ahead buffer }
Procedure IOSetRealRelTablet( state: boolean );  { if state is true, then
                                                   tablet will be a true
                                                   relative tablet }
Procedure IOChooseTablet( model : TabletType );  { choose which tablet will
                                                   be operating }
Procedure IOSetBitPadUpdateTimeOut( cnt: integer );  { set time-out constant
                                                       for BitPad updates }

procedure IOCursorMode( M: CursMode );
```

Abstract:

Sets the mode for the cursor. If the mode m is set to TrackCursor, Tablet coordinates are copied every 1/60th second into the cursor position. If it's set to IndepCursor, coordinates are changed only by the user. If it is set to OffCursor, no cursor is displayed.

Parameters:

m - the new mode for the cursor.

```
procedure IOSetModeTablet( M: TabletMode );
```

Abstract:

Sets the mode to tell what kind of tablet is currently in use.

Parameters:
m - the mode for the tablet.

```
procedure IOLoadCursor( Pat: CurPatPtr; pX, pY: integer );
```

Abstract:

Loads a user cursor pattern into the screen cursor.

Parameters:
Pat - a pointer to a cursor. It should be quad-word aligned.

pX and pY - offsets in the cursor where the origin is thought to be. For example, if the cursor is a bull's eye, 31 bits diameter flushed to the upper left corner of the cursor box, using (pX, pY) = (15, 15) will have the cursor surround the things pointed at.

NOTE: This procedure supports a cursor which is 56 x 64; with a scan line length of 4

```
procedure IOReadCursPicture( Pat: CurPatPtr; var pX, pY: integer );
```

Abstract:

Copies the current cursor picture into Pat and sets pX and pY with the offsets for the current cursor.

Parameters:

Pat, pX, and pY are filled with data on the current cursor. Note that Pat must be quad-word aligned.

```
Procedure IOSetFunction( f: CursFunction );
```

Abstract:

Sets the cursor function.

Parameters:

f - the function to set the cursor to.

```
procedure IOSetCursorPos( x, y: integer );
```

Abstract:

If the cursor's mode is not TrackCursor, this procedure sets the cursor's x and y positions. If tracking, it sets both tablet and cursor.

Parameters:

x and y - the new cursor coordinates.

```
procedure IOSetTabPos( x, y: integer );
```

Abstract:

If the cursor's mode is not set to TrackCursor, IOSetTabPos sets the tablet's x and y positions. If the mode is TrackCursor, both tablet and cursor are set.

Parameters:

x and y - the new tablet coordinates.

```
procedure IOReadTablet( var tabX, tabY: integer );
```

Abstract:

Reads tablet coordinates.

Parameters:

tabX and tabY - set to x and y values of the tablet.

```
Procedure IOScreenSize( newSize: Integer; Complement: Boolean );
```

Abstract:

Changes the amount of screen visible to the user (the rest is turned off, hence not displayed). The cursor is prevented from going into the undisplayed part of the screen.

Parameters:

newSize - number of scan lines in new screen; it must be a multiple of 128.

Complement - tells whether the rest of the screen should be the opposite color of the displayed part.

```
Procedure IOGetTime( var t : double );
```

Abstract:

Reads the 60 Hertz clock.

Parameters:
t - set to the new time.

```
procedure IOKeyDisable( var OldKeyEnable: Boolean );
```

Abstract:

IOKeyDisable is used to disable keyboard interrupts. This is used to delay processing of control-c, control-shift-c and control-shift-d at critical times. The old value of the keyboard interrupt enable is returned and must be passed back to IOKeyEnable when re-enabling keyboard interrupts. Characters typed while keyboard interrupts are disabled are remembered. When keyboard interrupts are re-enabled, the characters are processed.

Parameters:
OldKeyEnable - set to the old value of the enable.

```
procedure IOKeyEnable( OldKeyEnable: Boolean );
```

Abstract:

IOKeyEnable is used to enable keyboard interrupts. The old value of the keyboard interrupt enable (as returned from IOKeyDisable) must be passed to IOKeyEnable when re-enabling keyboard interrupts. If characters were typed while keyboard interrupts were enabled, IOKeyEnable calls KeyIntr to process those characters. The master interrupt control (INTON and INTOFF QCodes) must be on when this procedure is called.

Parameters:
OldKeyEnable - the old value of the enable.

```
procedure IOKeyClear;
```

Abstract:

IOKeyClear clears the keyboard type-ahead buffer.

Procedure IOSetRealRelTablet( state : boolean );

Abstract:

Allows the Kriz Tablet or GPIBBItPad to be handled in a true relative mode.

Parameters:

state - if true then a true relative mode is obtained whenever the TabMode = relTablet; lifting the mouse/puck/pen from the tablet surface and then returning it does not alter cursor position on the screen - only the movement of the mouse/puck/pen on the tablet surface will cause corresponding delta-x and delta-y changes in cursor position.

- if false, then x and y co-ords are simple linear transformation of the actual values to provide a 1-1 mapping of the 768 by 1024 screeninto the tabler surface whenever TabMode = relTablet.

Procedure IOChooseTablet( model: TabletType );

Abstract:

Allows selection of tablet that will provide co-ordinate positions and switch values.

Parameters:

model - choices are KrizTablet, GPIBBitPad, or NoPointDev

Side Effects: Depending upon the state of TabMode, we may have to call IOSetModeTablet to turn one tablet off and the other one on.

procedure IOSetBitPadUpdateTimeOut( Cnt: integer );

Abstract:

Set the time-out used to determine if the puck/pen is off the BitPad.

Parameters:

cnt - Cnt*1/60sec is the actual time-out.

```
module IO_Private;

     Abstract:

          IO type, and data definitions common to the IO system but not
          available to other Perq modules.

          Copyright (C) PERQ Systems Corporation, 1982, 1983

Version Number V6.2
{>>>>>>>>>>>>>>>>>>>>} exports {<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<}

imports SystemDefs from SystemDefs;
imports IO_Unit from IO_Unit;
imports IO_Others from IO_Others;
imports Raster from Raster;

const

  { Micro-code I/O Entry Points }

  { These entry points are parameters to the Pascal supported Q-code    }
  { "STARTIO". Additional parameters, where required, are documented    }
  { in the explainations. Note that ETOS refers to the top of the       }
  { expression stack as seen by the Pascal programmer. After a StartIO  }
  { instruction is executed, the I/O Entry Point has been pushed to the }
  { ETOS, and thus, what looks like ETOS to Pascal, is ETOS-1 to the    }
  { micro-code. Note also that all addresses are virtual addresses, and}
  { that }

    EP_IOStart = 0; { I/O Micro-Code Initialization Entry Point }
                    { Expects a pointer to the micro-code device}
                    { table at top of stack. Expects entry 0 of}
                    { the device table to have its data control }
                    { pointer set to the address of the         }
                    { interrupt vector. To be executed once     }
                    { at system boot time as part of startup.   }
                    { Also sets stack base and limit.           }

    EP_HardDisk = 1; { Hard Disk I/O request }
                     { Expects the hard disk uCode Device Table }
                     { entry's data control pointer to point to }
                     { a Disk Control Block. }

    EP_Ethernet = 2; { 3/10MB Ethernet I/O request }
                     { Expects the ethernet uCode Device Table  }
                     { entry's data control pointer to point to }
                     { an Ethernet Control Block. }

    EP_SetEnableMask = 3; { Update device's interrupt enable mask  }
                     { Expects a Z80 device ID at ETOS and a mask  }
                     { to be applied to interrupts for that       }
                     { device at ETOS-1. }

    EP_ReadTimer = 4;{ Returns a count of 'jiffies' since last }
```

```
                    { 'ReadTimer' request. Returns jiffies   }
                    { on ETOS, ETOS-1.  ETOS-2 = return code. }

    EP_ReadCause = 5;{ Determine the cause of an interrupt and }
                    { attention from a device. Expects a device ID
                    { at ETOS. Returns interrupt cause on ETOS, }
                    { attention cause on ETOS-1.  }

    EP_Z80Msg = 6;  { Enqueue a message for delivery to Z80 }
                    { Expects a pointer to a message at     }
                    { ETOS,ETOS-1 and a destination queue   }
                    { at ETOS-2. Returns a results indicator}
                    { (integer) at ETOS  }

    { StartIO entry points 7 and 8 no longer exists and can be
      redefined. }

    EP_UcodeMsg = 9; { Get a Z80 message from the Ucode sent message }
                    { queue.  The Ucode returns a pointer to a Z80  }
                    { message at ETOS, ETOS-1. }

    EP_GetChar = 10; { Return next char from the circular buffer of }
                    { the specified device, if any has been received, }
                    { with completion status.  Expects a device ID at }
                    { ETOS.  Returns a results indicator (integer) at  }
                    { ETOS and a byte at ETOS-1. }

    EP_PutCircBuffer = 11; { Place a byte onto a circular buffer }
                    { Expects a character on ETOS, a pointer to a     }
                    { circular buffer control block at ETOS-1,ETOS-2, }
                    { and returns a results indicator (integer) at ETOS. }


    EP_GetCircBuffer = 12; { Return next byte from Specified Circular }
                    { buffer(if not empty) with completion status.   }
                    { Expects a pointer to a circular buffer control }
                    { block at ETOS,ETOS-1. }
                    { Returns a results indicator (integer) at ETOS  }
                    { and a byte at ETOS-1. }

{ Device table definitions }

 { Device type: }

   Dev_Unused = 0;      { indicates that the device field is not used }

   { Hard Disk device types }
   Dev_Shugart = 1;     { Shugart disk drive }
   Dev_Micropolis = 2; { Micropolis disk drive }
   Dev_SMD = 3;         { Store Module Technology }

 { Intr cause - all numbers are indexex into the IntrCause field }

  { Interrupt causes common to all devices }

   Dev_Attention = 0;     { We received an attention msg. from Z80 }
```

```
    Dev_AckReceived = 1;  { Z80 'Acked' our request msg. }
    Dev_NakReceived = 2;  { Z80 'Naked' our request msg. }
    Dev_StatReceived = 3; { Z80 sent us a status msg. }

   { Intr cause - Device Specific }

    { the following is valid for RSB, RSA, GPIB, and the KeyBoard }

    Dev_DataAvailable = 5;    { Data has arrived in the circular buffer }

    { the following is valid for screen out }

    Dev_ScreenUpdate = 6;     { time for a screen update }

const

  CirBufSize = #100-3;

type

{ The definition of a circular buffer }

  CirBufItem = packed record   { one element of a circular buffer }
   ch: char;                   { the character }
   status: 0..255              { and a byte of status information }
   end;

  CircularBuffer = packed record    { the circular buffer for characters }
   Length: integer;            { number of characters in the buffer }
   RdPtr:  integer;            { where to get characters from }
   WrPtr:  integer;            { where to put characters to }
   Buffer: packed array[0..CirBufSize-1] of CirBufItem
   end;                        { lastly, the buffer of items }

  CirBufPtr = ^CircularBuffer;      { points to a circular buffer }

  Z_CmdRegister = packed record
  case integer of
    1: ( Bits : packed array [0..15] of boolean );
    2: ( Number : integer );
  end;

type

  DevTblEntry = packed record { Each entry must be }
                                                  { quad-word aligned. }
    IntrCause    : Z_CmdRegister;
                 { This is a bit map of the cause(s) of the interrupt(s)  }
                 { which the Pascal has not as yet received.              }

    IntrPriority : Z_CmdRegister;
                 { This array determines the interrupt vector entry and }
                 { thus the priority to be used when an interrupt for   }
                 { this device occurs.  0=highest, 15=lowest.           }

    EnableMask   : Z_CmdRegister;
```

```
                { This is a mask determining which types of interrupts,  }
                { as defined in the IntrCause array, will produce Pascal }
                { level interrupts.  '1' enables intr. at correponding   }
                { bit position in IntrCause, '0' masks the intr. until   }
                { the mask bit is again set to '1'}

    ATNCause    : Z_CmdRegister;
                { This is a device-specific bit map of the reason(s) for }
                { an attention message from the device.                  }

    pCirBuf     : CirBufPtr;              { KEEP THIS QUAD WORD ALIGNED }
                { Points to a Circular Buffer }

    pStatus     : pointer;
                { Points to device-specific device status information. }

    pDataCtrl   : pointer;                { KEEP THIS QUAD WORD ALIGNED }
                { Points to device-specific I/O control structure to be }
                { used in all operations not using the circular buffer }

    DeviceType  : integer;
                { Differentiates units as to their type.  This code will }
                { vary among different hard disk drives where the uCode  }
                { must process requests for the drives differently.      }

    Reserved    : packed array [0..0] of integer; { Filler to insure a }
                                                  { quad word alignment}
  end;

  pUDeviceTable = ^UDeviceTable;
  UDeviceTable = array [0..MaxUnit] of DevTblEntry;

{ The definition of a high volume data control buffer }

  HiVolBlock = packed record  { MUST BE QUAD WORD ALIGNED }
   DataByteCnt: integer;     { number of bytes in the buffer }
   pDataBuffer: IOBufPtr     { pointer to buffer supplied by the user }
   end;
  pHiVolBlock = ^HiVolBlock;

Type

      IOPtrKludge = record case integer of
                      1: (Buffer: IOBufPtr);
                      2: (Offset: Integer;
                          Segment: Integer)
                      end;

type

  Z_Commands = ( { The order of declaration of these commands }
                 { should not be changed as the uCode and Z80 }
                 { depend on their generated values remaining }
                 { constant. }
       Z_Illegal,          Z_RequestData,
       Z_BlockData,        Z_SendData,
```

```
          Z_ACK,              Z_NAK,
          Z_ATN,              Z_Status,
          Z_Seek,             Z_Config,
          Z_Boot,             Z_Reset,
          Z_Sense,            Z_WriteRegisters,
          Z_InHiVolumeStart,  Z_OutHiVolumeStart,
          Z_ReadData,         Z_WriteData,
          Z_ReadID,           Z_ReadDeletedData,
          Z_WrtDeletedData,   Z_Specify,
          Z_Format,           Z_Recal,
          Z_SenseDriveStatus, Z_EOIdata  );

const

  Z_SOM = 170 ;  { Start of Z80 message character for Z80 messages }
                 { Binary 10101010 to make it unlikely to match a  }
                 { device, command, or byte count. }

{ Length constants for data portion of Z80 messages }

  Z_NoData = 2;

  Z_FirstData = Z_NoData + 1;

  Z_MaxData = 14;

  Z_DataSize = Z_MaxData - Z_NoData;

type

  Z_Queue = (Z_Q0, Z_Q1, Z_Q2, Z_Q3 );

  Z_Data = packed array[Z_FirstData..Z_MaxData] of 0..255;

  pZ_Msg = ^Z_Msg;
  Z_Msg = packed record
    pNext        : pZ_Msg;
    UCodeArea    : long;   { micro code uses these two words, don't touch }
    SOMDelimiter : 0..255;
    ByteCount    : 0..255;
    Device       : 0..255;
    Command      : 0..255;
    Data         : Z_Data
    end;

  Z_MsgPtrKludge = record case boolean of
      true: (pMsg: pZ_Msg);
      false:(Offset: integer;
             Segment: integer);
      end;

const
  SLeftX = 0;              { left most X coordinate }
  SRightX = 767;           { right most X coordinate }
  STopY = 0;               { top most coordinate }
```

```
Var
  SBottomY : integer;      { current bottom most Y coordinate }

  pUDevTab : pUDeviceTable;   { points to the device table }

  Z_MsgNotAvailable : integer; { count of number of times Ucode did not
                               { have an empty Z80 message when asked
                                 for one }

  IOSegNum : integer;  { For debugging, holds the actual segment number }
                       { from which IOSegment data structures are       }
                       { allocated.  Holds actual segment number of the }
                       { IOSeg when not testing. }

  Z_IntDisabled : boolean;

  KeyEnabled : boolean;

  TimeBuf : ^long;     { points to timer information, here because }
                       { there is no IOTimer }

  RaiseException : packed array[0..MaxUnit] of packed record
                     Attention : boolean;
                     DataAvailable : boolean
                     end;

Procedure Z_SendMsg( pMsgToSend : pZ_Msg; SendQueue : Z_Queue );

function Z_DqSysMsg : pZ_Msg;

procedure Z_QSysMsg (pMessage : pZ_Msg );

procedure Z_CriticalSection ( BeginIt : boolean;
                              Unit    : integer;
                          var Save    : integer);

Procedure Z_SendMsg( pMsgToSend : pZ_Msg; SendQueue : Z_Queue );
```

Abstract:

This procedure is called by I/O system logic to send a message to the Z80 I/O controller. The message will be queued by the Perq micro-code to be sent when the Z80 port becomes available. Return to the caller is immediate; the completion of transmission will be detected from Z80 return messages.

Parameters:

pMsgToSend: The single parameter is a pointer to a record which contains the message as well as the queue to place the message on.

```
function Z_DqSysMsg : pZ_Msg;
```

Abstract:

This routine removes a Z80 message from system owned queue of messages which have been sent to the Z80. Each call to this routine will return with ether a pointer to a Z80 message.

This procedure is meant to be called by routines needing an empty Z80 message.

Returns:

The value returned is a pointer to a Z80 message. If no messages remain on the system owned queue this returned value is NIL.

```
procedure Z_QSysMsg ( pMessage : pZ_Msg);
```

Abstract:

This routine places a Z80 message onto the system owned queue of messages which have been sent to the Z80. Its purpose is to provide IO routines with system owned messages and to provide initialization logic with a way of first adding messages to the queue.

Parameters:

pMessage - Message to be sent to the Z80.

```
procedure Z_CriticalSection ( BeginIt : boolean; Unit  : integer; var Save
    : integer);
```

Abstract:

This routine implements a critical section by disabling all interrupts for a device and restoring the interrupt state on completion of the critical section.

Parameters:

BeginIt - When true, a critical section should begin, else the critical section is ended.

Unit - Unit whose interrupts will be enabled/disabled.

Save - When a critical section should begin, this value will be returned to the caller containing the interrupt mask before all interrupt types were disabled.

When the critical section is being ended, this value is an input parameter and holds(the previously saved) interrupt mask which should be retored.

```
module IO_Unit;

IO_Unit - Unit IO routines.

Copyright (C) 1982, 1983, PERQ Systems Corporation

     Abstract:

          IO_Unit exports constants, types, variables, and procedures needed
          to perform IO on the various IO Units, (devices).

          Design: All procedures call device dependent routines to do the
          actual IO

Version Number V7.11
{*****************************}  Exports
{*****************************}

imports SystemDefs  from SystemDefs;

const

  { Device Code Assignments for device table }

{ NOTE: The order of device declaration is important;  all Z80-controlled }
{       devices have been assigned a sub-range of contiguous values.  Be  }
{       sure to Check IO_Defs_Private before modifying these values!      }

        IOStart = 0;              { System Initialization }
        HardDisk = 1;  { Has a device table entry }

{$ifc Ether3MBaud then}
        Ether3 = 2;  { Has a device table entry }
{$elsec} {$ifc Ether10MBaud then}
        Ether10 = 2;  { Has a device table entry }
{$endc} {$endc}

        Floppy    =  3;
        RSA       =  4;  RS232Out = RSA; RS232In = RSA;
        RSB       =  5;  { valid for EIO boards only }
        Speech    =  6;
        GPIB      =  7;  GPIBIn = GPIB; GPIBOUT = GPIB;
        Keyboard  =  8;
        Timer     =  9;
        Clock     = 10;
        PointDev  = 11;  Tablet = PointDev;
        TransKey  = 12;
        ScreenOut = 13;
        EIODisk   = 14;
        Z80       = 15;

        LastUnit = Z80;         { for unit validity checking }
        MaxUnit = LastUnit;     { highest legal device code }

        RSExt = 0;          { RS-232 Speeds }
        RS110 = 1;
```

```
        RS150 = 2;
        RS300 = 3;
        RS600 = 4;
        RS1200 = 5;
        RS2400 = 6;
        RS4800 = 7;
        RS9600 = 8;
        RS19200 = 9;

        RS_MaxWords = 6;
        RS_MaxBytes = RS_MaxWords * 2;

type
        UnitRng = 0..MaxUnit;

        IOBufPtr = ^IOBuffer;
        IOBuffer = array[0..0] of integer;

        CBufPtr = ^CBufr;
        CBufr = packed array[0..0] of char;     { same as Memory, except }
                                                { for character buffers }
        BigStr = String[255];                   { A big String }

        IOStatPtr = ^IOStatus;
        IOStatus = record
                HardStatus: integer;            { hardware status return }
                SoftStatus: integer;            { device independent
                                                  status }
                BytesTransferred: integer
                end;

        IOCommands = (IOReset,      IORead,     IOWrite,      IOSeek,
                      IOFormat,     IODiagRead, IOWriteFirst, IOIdle,
                      IOWriteEOI,   IOConfigure,IOWriteRegs,  IOSense,
                      IOWriteHiVol, IOReadHiVol,IOReadID,     IOFlush );

        IOHeadPtr = ^IOHeader;
        IOHeader = record               { Hard disk header record }
             SerialNum : double;        { Serial number of the file }
             LogBlock  : integer;       { The logical block number }
             Filler    : integer;
             NextAdr   : double;        { Address of next block in the
                                          file }
             PrevAdr   : double         { Address of previous block }
             end;

       IOIntrTypes = (IOATNInterrupt, IODataInterrupt);

    { the following is useful for an IOWriteReg command to the RS232 }

    RS_WrtReg = packed record    { Write to Chip registers }
        ID : 0..255;
        case integer of
          0 : { Write to command register }
              (NextRegisterPointer :  0..7;
               Command  : (R_NullCommand,      R_SendAbort,
```

```
      pClockStat = ^ClockStat;
      ClockStat = packed record     { IOSense to the Clock provides this }
           Cycles  : 0..255;
           Year    : 0..255;
           Month   : 0..255;
           Day     : 0..255;
           Hour    : 0..255;
           Minute  : 0..255;
           Second  : 0..255;
           Jiffies : 0..255
           end;

      pZ80Stat = ^Z80Stat;
      Z80Stat = packed record          { IOSense to the Z80 provides the }
           MajorVersionNum : 0..255; { version number of the code running }
           MinorVersionNum : 0..255; { on the Z80, interpret as }
           end;                      { Version Major.Minor }

      pPointDevStat = ^PointDevStat;
      PointDevStat = packed record    { zero means PointDev configured off }
           OnOff : 0..255;            { nonzero means PointDev configured
                                        on }
           end;

      pKeyStat = ^IO_KeyStat;
      IO_KeyStat  = packed record
           OnOff    : 0..255;        { Zero means keyboard is configured
                                       off }
           OverFlow : 0..255;        { nonzero means overrun on in buffers }
           end;


RS_StatusType = (RS_Config, RS_PStat, RS_GStat,
                 RS_MapBytes, RS_MapWords);

pRS232Stat = ^RS232Stat;
RS232Stat = packed record
  case RS_StatusType of
    RS_MapBytes: ( Byte : packed array [ 1..RS_MaxBytes] of 0..255 );
    RS_MapWords: ( Word : packed array [ 1..RS_MaxWords] of integer );
    RS_Config  : ( XmitRate : 0..255;
                   RcvRate : 0..255
                 );
    RS_PStat   : ( Reg : packed array [1..6] of RS_WrtReg );
    RS_GStat   : { Read General Status - SIO Chip's Read Register #0 }
                 ( RxCharAvailable  : boolean;
                   IntPending       : boolean;
                   TxBufferEmpty    : boolean;
                   DCD              : boolean;
                   SyncHunt         : boolean;
                   CTS              : boolean;
                   TransmitUnderRun : boolean;
                   BreakAbort       : boolean;
                   { Read Special Condition - SIO Chip's Read Register #1 }
                   AllSent          : boolean;
```

```
                Residue         : 0..7;
                ParityError     : boolean;
                RxOverRun       : boolean;
                CrcFramingError : boolean;
                EndOfFrame      : boolean );
    end;

 { Use of this status block is discouraged,  Use the IOSense, IOConfigure,
   and IOWriteRegs command with UnitIO instead of IOPutStatus and
   IOGetStatus }

  DevStatusBlock = packed record
    ByteCnt: integer;       { # of status bytes }
    case UnitRng of
      KeyBoard,
{$ifc Ether3MBaud then}
      Ether3,
{$endc}
      Clock: (DevEnable: boolean);

      Tablet: (OldStanleyEnable : boolean;  { should always be false }
               KrizEnable : boolean;
               case boolean of
                 true  : { GET STATUS }
                         ( TAbFill : 0..#77;
                           TabOverRun : 0..255 );
                 false : { PUT STATUS }
                         ( { nothing } ));

      RS232In,
      RS232Out: (RSRcvEnable : boolean;
                 RSFill      : 0..127;
                 RSSpeed     : 0..255;
                 RSParity    : (NoParity, OddParity, IllegParity,
                                EvenParity);
                 RSStopBits  : (Syncr,Stop1,Stop1x5,Stop2);
                 RSXmitBits  : (Send5,Send7,Send6,Send8);
                 RSRcvBits   : (Rcv5,Rcv7,Rcv6,Rcv8));

      Floppy: (case integer of { Get or Put }
                 1: { GET STATUS }
                    (FlpUnit      : 0..3;
                     FlpHead      : 0..1;
                     FlpNotReady  : boolean;
                     FlpEquipChk  : boolean;
                     FlpSeekEnd   : boolean;
                     FlpIntrCode  : 0..3;
                     case integer of
                       1 {IORead, IOWrite, IOFormat}:
                         (FlpMissAddr    : boolean; { in data or header }
                          FlpNotWritable : boolean;
                          FlpNoData      : boolean;
                          FlpFill1       : 0..1;
                          FlpOverrun     : boolean;
                          FlpDataError   : boolean; { in data or header }
                          FlpFill2       : 0..1;
```

```
                        FlpEndCylinder : boolean;
                        FlpDataMissAddr: boolean; { in data }
                        FlpBadCylinder : boolean;
                        FlpFill3       : 0..3;
                        FlpWrongCylinder : boolean;
                        FlpDataDataError : boolean; { in data }
                        FlpFill4       : 0..3;
                        FlpCylinderByte  : 0..255;
                        FlpHeadByte      : 0..255;
                        FlpSectorByte    : 0..255;
                        FlpSizeSectorByte: 0..255 );
                    2 {IOSeek}:
                      (FlpPresentCylinder: 0..255));
                2: { PUT STATUS }
                   (FlpDensity : 0..255;   { single = 0, double = #100 }
                    FlpHeads   : 0..255;   { 1 or 2 heads }
                    FlpEnable  : boolean );
                3: { BYTE ACCESS }
                   (FlpByte1 : 0..255;
                    FlpByte2 : 0..255;
                    FlpByte3 : 0..255;
                    FlpByte4 : 0..255;
                    FlpByte5 : 0..255;
                    FlpByte6 : 0..255;
                    FlpByte7 : 0..255 )  );

    GPIBIn,
    GPIBOut: { GET STATUS ONLY }
             (Int1Status  : 0..255;
              AddrSwitch  : 0..255;
              AddrStatus  : 0..255;
              CmdPassThru : 0..255;
              Int0Status  : 0..255;
              BusStatus   : 0..255 )

    end;

{ hard status type information }

  DskResult = packed record
    case integer of
      0  : ( Result : integer );
      1 : ( Cntlerror : ( DskOK,
                          AddrsErr,          { address error }
                          PHCRC,             { Physical Header CRC }
                          LHSer,             { Logical Serial Wrong }
                          LHLB,              { Logical Block Wrong }
                          LHCRC,             { Logical Header CRC }
                          DaCRC,             { Data CRC }
                          Busy);
            Fill2     : boolean;
            TrackZero : boolean;
            WriteFault : boolean;
            SeekComplete : boolean;
            DriveReady : boolean);
      2 :  {for cio micropolis disks}
```

```
            ( CioMCntlError: ( CioMDskOK,
                               CioMAddrsErr,        { address error }
                               CioMPHCRC,           { Physical Header CRC }
                               CioMLHSer,           { Logical Serial Wrong }
                               CioMLHLB,            { Logical Block Wrong }
                               CioMLHCRC,           { Logical Header CRC }
                               CioMDaCRC,           { Data CRC }
                               CioMBusy);
                  CioMIndex      : boolean;
                  CioMIllegalAddr : boolean;
                  CioMFault : boolean;
                  CioMSeekComplete : boolean;
                  CioMDriveReady : boolean)
        end;
{ Floppy special data descriptions }
{ This information is returned from IOReadID for the floppy }

  FlpPhyHdr = Packed record       { used for IOFormat and IOReadID }
              Cylinder : 0..255; { cylinder number }
              Head : 0..255;     { Head number }
              Sector : 0..255;   { Sector number }
              Size : 0..255;     { Size (0=128 bytes, 1=256 bytes }
              end;

  pFlpPhyHdr = ^FlpPhyHdr;        { Recast to IOBufPtr for IOReadID }

{ This information is passed to IOFormat and specifies how to format }

  FlpFmtHdrs = Array [1..26] of FlpPhyHdr; { input to IOFormat command }
  pFlpFmtHdrs = ^FlpFmtHdrs;          { RECAST to IOBufPtr for IOFormat }

Var
       CtrlSPending : boolean;  { True: Control S has halted screen
                                  output }
       IOInProgress: boolean;   { false when speech is active }
       IO24MByte : boolean;        { true if the disk is 24 MBytes }

Exception DevInterrupt(Unit: UnitRng; IntType: IOIntrTypes; ATNCause:
    Integer);
Function IOCRead(Unit: UnitRng; var Ch: char): integer;
Function IOCWrite(Unit: UnitRng; Ch: char): integer;
Procedure UnitIO(Unit: UnitRng;
                 Bufr: IOBufPtr;
                 Command: IOCommands;
                 ByteCnt: integer;
                 LogAdr: double;
                 HdPtr: IOHeadPtr;
                 StsPtr: IOStatPtr);
Procedure IOWait(var Stats: IOStatus);
Function IOBusy(var Stats: IOStatus): boolean;
Procedure IOPutStatus(Unit: UnitRng; var StatBlk: DevStatusBlock);
Procedure IOGetStatus(Unit: UnitRng; var StatBlk: DevStatusBlock);
Procedure IOBeep;
Function IOCRNext( Unit: UnitRng; Var Ch: char ): integer;
Function IOCPresent( Unit: UnitRng ): boolean;
Procedure IOClearExceptions;
```

```
Procedure IOSetExceptions( Unit : UnitRng;
                           IntType : IOIntrTypes;
                           var Setting:boolean);

{$ifc Ether3MBaud then}
Function Ether3Transmit(Buff: IOBufPtr; WdCnt: integer) : integer;

Function Ether3Receive(Buff: IOBufPtr; var WdCnt: integer; timeout: integer)
                       : integer;

Function Ether3Start(Promiscuous, Restart: boolean) : integer;
{$endc}

Function IOCRead(Unit: UnitRng; var Ch: char): integer;
```

Abstract:

Reads a character from a character device and returns a completion or error code.

Parameters:

Unit - device from which to read the character.

Ch - character to read.

Returns:

A condition code as defined in the module IOErrors.

```
Function IOCWrite( Unit: UnitRng; Ch: char):integer;
```

Abstract:

Writes a character to a character device and returns a completion or error code. Delays if the buffer is full. Returns an error if the condition doesn't clear up.

Parameters:

Unit - device onto which the character will be written.

Ch - character to write.

Returns:

Condition code as defined by the module IOErrors.

```
Procedure IOWait( var Stats: IOStatus );

     Abstract:

          Hangs until an IO operation inititated by UnitIO is complete.

     Parameters:
          Stats - Status block that was given to UnitIO when the operation
               was initiated.

Function IOBusy( var Stats: IOStatus): boolean;

     Abstract:

          Determines whether or not I/O is complete.

     Parameters:
          Stats - Status block that was  given  to  UnitIO when the
               operation was initiated.

     Returns:

          True if IO is not complete, false if it is.

Procedure IOPutStatus( Unit: UnitRng; var StatBlk:DevStatusBlock );

     Abstract:

          Sets device's characteristics.  Has no effect if the device has no
          settable status.

     Parameters:
          Unit - device whose characteristics  are  to  be set.

          StatBlk - block containing characteristics to be set.

Procedure IOGetStatus( Unit: UnitRng; var StatBlk:DevStatusBlock );

     Abstract:

          Reads device status.  Has no effect if the device has no readable
          status.

     Parameters:
          Unit  -  device whose characteristics are to be read.

          StatBlk - block to which device status is to  be returned.
```

```
Procedure UnitIO( Unit: UnitRng; Bufr: IOBufPtr; Command: IOCommands;
     ByteCnt: integer; LogAdr: double; HdPtr: IOHeadPtr; StsPtr: IOStatPtr
     );
```

Abstract:

IO to non-character devices.

Parameters:

Unit - the device.

Bufr - buffer for data transfers, if requested.

Command - operation to be performed on the device.

ByteCnt - number of bytes to be transferred.

LogAdr - logical address for block structured devices.

HdPtr - pointer to the logical header for operations with the hard disk.

StsPtr - resultant status from the operation.

```
procedure IOBeep;
```

Abstract:

Causes the PERQ to beep.

```
Function IOCRNext( Unit: UnitRng; Var Ch: char ): integer;
```

Abstract:

Reads a character from a character device and returns a completion or error code. We will always return with an error or a character.

Parameters:

Unit - device from which the character will be read
Ch - character read from device

Returns: condition code as defined by module IOErrors.

```
Function IOCPresent( Unit: UnitRng ): boolean;
```

Abstract:

Returns true if the Unit is a character device and has a character available for reading. Otherwise it returns false. It does not read the character.

Parameters:

Unit is the device to check for parameters

Returns: true if a character is available.

```
Procedure IOClearExceptions;
```

Abstract:

Disable the raising of exceptions for device interrupts. All devices are affected. Exception raising disabled is the normal case. This procedure also resets the enable mask of certain devices.

```
procedure IOSetExceptions( Unit : UnitRng;
        IntType : IOIntrTypes;
        var Setting:boolean);
```

Abstract:

Disables or enables the raising of an exception when the specified device raises the specified interrupt. Setting = true enables, Setting = false disables. Upon return, Setting will be set to to true if the raising of the specified exception was previously enabled, false if not.

Parameters:

Unit - device to enable/disable exceptions on
IntType - type of interrupts to enable/disable
Setting - true to enable, false to disable

Returns: setting returns previous state of exceptions.

```
module Lights;

Lights - Perq Lights.
J. P. Strait  26 May 81.
Copyright (C) PERQ Systems Corporation, 1981

    Abstract:

        This module defines the screen coordinates and size of the Perq
        "lights".  These are portions of the screen that are inverted
        during tedious operations such as recalibrating the disk and
        scavenging files (in FileAccess).

        Design: The lights must *not* extend below the 128th line of the
        screen.  The Y + Size must be less than or equal to 256.  It is a
        good idea for the lights to be totally inside of the title line.
        The current lights start at the left leave lots of room for new
        lights to the right of the current one.  There is room for 10
        lights all together

Version Number V1.4

exports

const

    LightUsed        = TRUE; {whether should use the lights at all}
    LightY           = 3;
    LightHeight      = 14;
    LightWidth       = 18;
    LightSpacing     = 3*LightWidth;
    LightRecalibrate = LightSpacing;
    LightScavenge    = LightRecalibrate + LightWidth + LightSpacing;
    LightSwap        = LightScavenge + LightWidth + LightSpacing;
    LightHardCopy    = LightSwap + LightWidth + LightSpacing;
```

```
module Loader;

Loader - Perq system loader.
J. P. Strait    10 Feb 81.  rewritten as a module.
Copyright (C) PERQ Systems Corporation, 1981, 1982.

     Abstract:

          This module implements the Perq POS system loader.  Given a
          run-file name as input, it loads and executes that program.  When
          the program terminates (normally or abnormally) it returns to the
          loader which returns to its caller.

Version Number V3.1


exports

   const LoaderVersion = '3.0';

procedure Load( RunFileName: String );

procedure Load( RunFileName: String );

     Abstract:

          Given a run-file name as input, this procedure loads and executes
          that program.  When the program terminates (normally or
          abnormally) it returns to the loader which returns to its caller.

     Parameters:
          RunFileName - Name of the .RUN file to load.  ".RUN" is appended
               if it is not already present.
```

```
module LoadZ80;

     Abstract:

          Module to read Tektronix format load files and load them into the
          Z80.

Version Number V0.3
{**************************} Exports {**************************}

Type
    TekResult = (TekOk, TekFNF, TekFMT, TekIO);

Function TekLoad(FileName : String; Var StartAddress : Integer) : TekResult;

Function TekLoad(FileName : String; Var StartAddress : Integer) : TekResult;

     Var
     Addr : Integer;          { Address of current block }
     TekFile : Packed File of 0..255;  { Input Tekhex format file }
     Binary : pByteArray;     { Binary data to be sent to Z80 }
     Data : tByteArray;       { data read from input file }
     i,j,k : integer;         { index }
     LogAdr : double;         { double word address for UnitIO }
     StsPtr : IOStatPtr;      { status for UnitIO call }
```

```
module Memory;

Memory - Perq memory manager.
J. P. Strait        1 Jan 80.
Copyright (C) PERQ Systems Corporation, 1980, 1982.

     Abstract:

          Memory is the Perq memory manager.  It supervises the segment
          tables and exports procedures for manipulating memory segments.
          Perq physical memory is segmented into separately addressable
          items (called segments) which may contain either code or data.

          Design: See the Q-Code reference manual.

Version Number V2.20

exports

const MemoryVersion = '2.18';

imports SystemDefs from SystemDefs;
imports Code from Code;

const SATSeg = 1;                { SAT segment }
      SITSeg = 2;                { SIT segement }
      FontSeg = 3;               { font segment }
      ScreenSeg = 4;             { screen segment }
      CursorSeg = 5;             { cursor segment }
      IOSeg = 6;                 { IO segment }
      SysNameSeg = 7;            { system segment names }

      BlocksInQuarterMeg = #1000; { 512 blocks in 1/4 meg memory}
      BlocksInHalfMeg    = #2000; { 1024 blocks in 1/2 meg memory}
      BlocksInMeg        = #4000; { 2048 blocks in 1 meg memory}

      BlocksForPortraitScreen = 192; { number of blocks needed for its
                                       segment}
      BlocksForLandscapeScreen = 320;

      MaxSegment = #137;         { should be 2**16 - 1 }

      SetStkBase = #60;
      SetStkLimit = #120;

{$ifc Ether3MBaud then}
      IOSegSize = 10;             { number of blocks in the IOSeg }
{$elsec}
{$ifc Ether10MBaud then}
      IOSegSize = 8;             { number of blocks in the IOSeg }
{$elsec}
      IOSegSize = 8;             { number of blocks in the IOSeg }
{$endc}
{$endc}

      SysSegLength = 8;         { length of name of a boot-loaded segment }
```

```
      MMMaxBlocks = #10000;   { maximum number of blocks in a segment }
      MMMaxCount =  #377;     { maximum reference count }
      MMMaxIntSize = MMMaxBlocks-1;
      MMMaxExtSize = MMMaxBlocks;

type MMBit4 = 0..#17;
     MMBit8 = 0..#377;
     MMBit12 = 0..#7777;
     MMIntSize = 0..MMMaxIntSize;
     MMExtSize = 1..MMMaxExtSize;
     MMAddress = integer;
     MMPosition = (MMLowPos, MMHighPos);

     SegmentNumber = integer;

     SegmentKind = (CodeSegment, DataSegment);

     SegmentMobility = (UnMovable, UnSwappable, LessSwappable, Swappable);

     MMFreeNode = record
      N: MMAddress;
      L: integer
      end;

     MMBlockArray = array[0..0] of array[0..127] of integer;

     pMMBlockArray = ^MMBlockArray;

     MMArray = record case Integer of
                1: (m: array[0..0] of MMFreeNode);
                2: (w: array[0..0] of Integer)
                end;

     pMMArray = ^MMArray;

     MMPointer = record case integer of
      1: (P: ^integer);
      2: (B: pMMBlockArray);
      3: (M: pMMArray);
      4: (Offset: MMAddress;
          Segmen: SegmentNumber)
      end;

     SATentry = packed record { Segment Address Table }
      { **** ENTRIES MUST BE TWO WORDS LONG **** }
      NotResident : boolean;              { 001 }
      Moving      : boolean;              { 002 }
      RecentlyUsed: boolean;              { 004 }
      Heap        : boolean;              { 010 }
      Kind        : SegmentKind;          { 020 }
      Full        : boolean;              { 040 }
      InUse       : boolean;              { 100 }
      Lost        : boolean;  { *** }     { 200 }
      BaseLower   : MMBit8;
      BaseUpper   : MMBit4;
```

```
      Size        : MMBit12
      end;

{$IFC WordSize(SATentry) <> 2 then}
{$MESSAGE ******* ERROR ***** SAT WRONG SIZE ****** }
{$ENDC}

     SITentry = packed record case integer of { Segment Information Table }
      { **** ENTRIES MUST BE EIGHT WORDS LONG **** }
      1: { real SIT entry }
         (NextSeg    : SegmentNumber;
          RefCount   : 0..MMMaxCount;
          IOCount    : 0..MMMaxCount;
          Mobility   : SegmentMobility;
          BootLoaded : Boolean;
          Increment  : MMIntSize; {used only if DataSegment and Heap is
                                   false}
          SwapInfo   : record case {BootLoaded:} Boolean of
                         True:  (BootLowerAddress: Integer;
                                 BootUpperAddress: Integer;
                                 BootLogBlock: Integer);
                         False: (DiskLowerAddress: Integer;
                                 DiskUpperAddress: Integer;
                                 DiskId: Integer)
                       end;
          case SegmentKind of
            DataSegment: (case {Heap:} Boolean of
                            False: (Maximum    : MMIntSize;
                                    Freelist   : MMAddress);
                            True:  (HeapNext   : SegmentNumber;
                                    Freelst    : MMAddress)
                          );
            CodeSegment: (Update     : TimeStamp)
         );
      2: { boot time information }
         (BootBlock: record
            CS: SegmentNumber;  { initial code segment }
            SS: SegmentNumber;  { initial stack segment }
            XX: Integer;        { used as interface between SysB and Config}
            VN: Integer;        { system version number }
            FF: SegmentNumber;  { first free segment number }
            FC: SegmentNumber;  { first system code segment }
            DK: integer;        { disk system was booted from }
            CH: integer         { char used in booting }
            end)
      end;

{$IFC WordSize(SITentry) <> 8 then}
{$MESSAGE ******* ERROR ***** SIT WRONG SIZE ****** }
{$ENDC}

     SATarray = array[0..0] of SATentry;

     SITarray = array[0..0] of SITentry;

     pSAT = ^SATarray;
```

```
    pSIT = ^SITarray;

    MMEdge = record
             H: SegmentNumber;  { Head }
             T: SegmentNumber   { Tail }
             end;

    SysSegName = packed array[1..SysSegLength] of Char;

    pSysNames = ^SysNameArray;

    SysNameArray = array[0..0] of SysSegName;

procedure InitMemory;
procedure DataSeg( var S: SegmentNumber );
procedure CodeOrDataSeg( var S: SegmentNumber );
procedure ChangeSize( S: SegmentNumber; Fsize: MMExtSize );
procedure CreateSegment( var S: SegmentNumber;
                         Fsize, Fincrement, Fmaximum: MMExtSize );
procedure IncRefCount( S: SegmentNumber );
procedure SetMobility( S: SegmentNumber; M: SegmentMobility );
procedure DecRefCount( S: SegmentNumber );
procedure SetIncrement( S: SegmentNumber; V: MMExtSize );
procedure SetMaximum( S: SegmentNumber; V: MMExtSize );
procedure SetHeap( S: SegmentNumber; V: boolean );
procedure SetKind( S: SegmentNumber; V: SegmentKind );
procedure MarkMemory;
procedure CleanUpMemory;
procedure FindCodeSegment( var S: SegmentNumber; Hint: SegHint );
procedure EnableSwapping( Where: Integer );
procedure DisableSwapping;
function  CurrentSegment: SegmentNumber;

exception UnusedSegment( S: SegmentNumber );
```

Abstract:

UnusedSegment is raised when the memory manager encounters a segment number which references a segment which is not in use. This may mean that a bad segment number was passed to some memory manager routine or that a bad address was de-referenced.

Parameters:

S - Segment number of the unused segment.

```
exception NotDataSegment( S: SegmentNumber );
```

Abstract:

NotDataSegment is raised when the number of a code segment is passed to some memory manager routine that requires the number of a data segment.

Parameters:

S - Segment number of the code segment.

```
exception BadSize( S: SegmentNumber; Fsize: Integer );
```

Abstract:

BadSize is raised when a bad Size value is passed to some memory manager routine. This usually means that the size passed to CreateSegment or ChangeSize is greater than the maximum size or less than one.

Parameters:

Fsize - The bad Size value.

```
exception BadIncrement( S: SegmentNumber; Fincrement: Integer );
```

Abstract:

BadIncrement is raised when a bad Increment value is passed to some memory manager routine. This usually means that the increment passed to CreateSegment is greater than 256 or less than one.

Parameters:

Fincrement - The bad Increment value.

```
exception BadMaximum( S: SegmentNumber; Fmaximum: Integer );
```

Abstract:

BadMaximum is raised when a bad Maximum value is passed to some memory manager routine. This usually means that the maximum passed to CreateSegment is greater than 256 or less than one.

Parameters:

Fmaximum - The bad Maximum value.

```
exception FullMemory;
```

Abstract:

FullMemory is raised when there is not enough physical memory to satisfy some memory manager request. This is raised only after swapping segments out and compacting memory.

```
exception CantMoveSegment( S: SegmentNumber );
```

Abstract:

CantMoveSegment is raised when the memory manager attempts to move a segment which is UnMovable or has a non-zero IO count.

Parameters:

S - The number of the segment which cannot be moved.

```
exception PartNotMounted;
```

Abstract:

PartNotMounted is raised when 1) the memory manager attempts to swap a data segment out for the first time; and 2) the partition which is to be used for swapping is no longer mounted.

```
exception SwapInFailure( S: SegmentNumber );
```

Abstract:

SwapInFailure is raised when the swap file cannot be found for a segment which is marked as swapped out. This is an error which should never happen in a debugged system. It usually means that there is a bug in the memory manager or that the segment tables have been clobbered.

Parameters:

S - The number of the segment which could not be swapped in.

```
exception EdgeFailure;
```

Abstract:

EdgeFailure is raised by MakeEdge when it discovers that the SIT entries are not linked together into a circular list. This is an error which should never happen in a debugged system. It usually means that there is a bug in the memory manager or that the segment tables have been clobbered.

```
exception NilPointer;
```

Abstract:

NilPointer is raised when a Nil pointer is used or passed to Dispose.

```
exception BadPointer;
```

Abstract:

BadPointer is raised when a bad pointer is passed to Dispose.

Parameters:

```
exception FullSegment;
```

Abstract:

FullSegment is raised by New when it discovers that there is not enough room to allocate and the segment cannot be enlarged (its size has reached its maximum).

```
exception NoFreeSegments;
```

Abstract:

NoFreeSegments is raised when the memory manager discovers that all of the segment numbers are in use and it needs another one. This is equivalent to "Segment table full".

```
exception SwapError;
```

Abstract:

SwapError is raised if the one of the memory managers swapping routines is called when swapping is disabled. This is an error which should never happen in a debugged system. It usually means that there is a bug in the memory manager.

```
var SAT: pSAT;
    SIT: pSIT;
    MMFirst, MMFree, MMLast, MMHeap: SegmentNumber;
    MMHole: MMEdge;
    MMState: (MMScan1, MMScan2, MMScan3, MMScan4, MMScan5,
              MMScan6, MMScan7, MMScan8, MMScan9, MMScan10,
              MMScan11,
              MMNotFound, MMFound);
    StackSegment: SegmentNumber;
    FirstSystemSeg: SegmentNumber;
```

```
        BootFileId: Integer;
        SwappingAllowed: Boolean;
        SwapId: Integer;
        MemoryInBlocks: Integer;  { amount of memory on this machine }

procedure InitMemory;
```

Abstract:

InitMemory initializes the memory manager. It is called once at system initialization and may not be called again. If the system was booted from a floppy, the system segments are all marked as UnSwappable.

```
procedure DataSeg( var S: SegmentNumber );
```

Abstract:

DataSeg is used to

1) - Determine if a given segment number represents a data segment.
2) - Find the default heap segment (in the case of an input parameter
of zero).

Parameters:

S - Data segment number--zero means the default heap segment.

Errors:

UnusedSegment - if S is not in use.
NotDataSegment - if S is not a data segment.

```
procedure CodeOrDataSeg( var S: SegmentNumber );
```

Abstract:

CodeOrDataSeg is used to

1) - Determine if a given segment number represents a defined segment
2) - Find the default heap segment (in the case of an input parameter
of zero).

Parameters:

S - Data segment number--zero means the default heap segment.

Errors: UnusedSegment if S is not in use.

procedure ChangeSize( S: SegmentNumber; Fsize: MMExtSize );

Abstract:

ChangeSize is used to change the size of an existing data segment.

Parameters:

S - Number of the segment whose size is to be changed.
Fsize - New size of the segment.

Errors:
UnusedSegment - if S is not in use.
BadSize - if Fsize is greater than the maximum size of S or less than one.
FullMemory - if there is not enough physical memory to increase the size of S.
CantMoveSegment - if the segment must be moved, but it is not movable or its IOCount is not zero.

procedure CreateSegment( var S: SegmentNumber; Fsize, Fincrement, Fmaximum: MMExtSize );

Abstract:

CreateSegment is used to create a new data segment. The size, increment and maximum may be up to 4096 blocks but clearly the size will be limited by the available memory. In addition, IT IS NOT POSSIBLE TO USE SEGMENTS BIGGER THAN 256 BLOCKS FOR NEW, DISPOSE, OR MAKEPTR. Therefore, the only way to address segments bigger than 256 blocks is using RasterOp.

Parameters:
S - Set to the number of the new segment.
Fsize - Desired size of the new segment in blocks.
Fincrement - Increment size of the new segment in blocks.
Fmaximum - Maximum size of the new segment.

Errors:
BadSize - if Fsize is greater than Fmaximum or less than one.
BadIncrement - if Fincrement is greater than MMMaxExtSize or less than one.
BadMaximum - if Fmaximum is greater than MMMaxExtSize or less than one.
FullMemory - if there is not enough physical memory to create the segment.

```
procedure IncRefCount( S: SegmentNumber );

    Abstract:

        IncRefCount increments the number of references to a segment. A
        non-zero reference count prevents a segment from being destroyed.
        A reference count greater than one indicates a system segment. If
        S is a heap, all segments in the heap are incremented.

    Parameters:

        S - Number of the segment.

    Errors: UnusedSegment  if S is not in use.

procedure SetMobility( S: SegmentNumber; M: SegmentMobility );

    Abstract:

        SetMobility sets the Mobility of a segment. The mobility may be
        set to one of the following values:

        Swappable       - segment is a candidate for swapping or moving.
        LessSwappable   - segment is a candidate for swapping or moving,
            but the memory manager will be more reluctant to swap.
        UnSwappable     - segment may not be swapped, but may be moved.
        UnMovable       - segment may not be swapped or moved. The
            RecentlyUsed bit of the segment is cleared also. Thus to
            make a segment a candidate for swapping, set its mobility to
            Swappable (even if it already swappable).

    Parameters:
        S - Segment number.
        M - Mobility.

    Errors:
        UnusedSegment  - if S is not in use.
        CantMoveSegment  - if the segment is changing from Swappable to
            UnMovabl an attempt is made to move the segment to the high
            end of memory. If it has a non-zero IO count this error is
            issued.
        FullMemory  - if the segment is changing from Swappable to
            UnSwappable, it is swapped out, and there isn't enough memory
            to swap it in.

procedure DecRefCount( S: SegmentNumber );

    Abstract:

        DecRefCount decrements the reference count of a segment by one.
        If reference and IO counts both become zero:
        - if the segment is a data segment, it is destroyed.
        - if the segment is a code segment, it is destroyed only if it is
            in the screen or is non-resident. If S is a heap, all
            segments in the heap are decremented.
```

Parameters:

S - Number of the segment.

Errors: UnusedSegment if S is not in use.

```
procedure SetIncrement( S: SegmentNumber; V: MMExtSize );
```

Abstract:

SetIncrement changes the increment size of a data segment.

Parameters:
S - Number of the segment.
V - New increment size.

Errors:
UnusedSegment - if S is not in use.
NotDataSegment - if S is not a data segment.
BadIncrement - if V is greater than MMMaxExtSize or less than one.

```
procedure SetMaximum( S: SegmentNumber; V: MMExtSize );
```

Abstract:

SetMaximum changes the maximum size of a data segment.

Parameters:
S - Number of the segment.
V - New maximum size.

Errors:
UnusedSegment - if S is not in use.
NotDataSegment - if S is not a data segment.
BadMaximum - if V is greater than MMMaxExtSize or less than one.

```
procedure SetHeap( S: SegmentNumber; V: boolean );
```

Abstract:

SetHeap changes the "Heap" attribute of a segment. This should not normally be changed by anyone other than Dynamic.

Parameters:
S - Number of the segment.
V - New value of the "Heap" attribute.

Errors:

UnusedSegment - if S is not in use.

```
procedure SetKind( S: SegmentNumber; V: SegmentKind );
```

Abstract:

SetKind changes the kind (code or data) of a segment.

Parameters:

S - Number of the segment.
V - New kind of the segment.

Errors: UnusedSegment - if S is not in use.

```
procedure MarkMemory;
```

Abstract:

MarkMemory marks all currently in use segments as system segments usually before loading a user program) by incrementing their reference counts.

```
procedure CleanUpMemory;
```

Abstract:

CleanUpMemory destroys all user segments (usually at the end of a program execution) by deecrementing the reference count of all segments.

```
procedure EnableSwapping( Where: Integer );
```

Abstract:

EnableSwapping turns the swapping system on, determines where swap files should be created, and locates the boot file.

Parameters:
Where - FileId of some file in the partition to be used for swap files.

```
procedure DisableSwapping;
```

Abstract:

DisableSwapping attempts to swap in all segments which are swapped out and then turns the swapping system off. If there is not enough physical memory to swap all segments in, swapping is not disabled.

Errors:
FullMemory - if there isn't enough memory to swap all segments in.

```
procedure FindCodeSegment( var S: SegmentNumber; Hint: SegHint );

    Abstract:

        FindCodeSegment searches for a code segment in the segment table
        which has a certain SegHint.  If such a segment is found, its
        RefCount is incremented and the segment number is returned.
        Otherwise, a zero segment number is returned.

        Segments which

        1) Have a DiskId equal to Hint.FId,
        2) Have an Update date/time not equal to Hint.Update, and
        3) Have a RefCount of zero

        are deleted.  This is done because such a segments reference code
        files which have been overwritten and are no longer valid.  Such
        segments will not get the memory manager into trouble, but they
        will never be used again, and it is just as well to get rid of
        them.

        **** Hint must be a valid hint.  That is, the file specified
        **** by Hint.FId must have a FileWriteDate equal to Hint.Update.

    Parameters:
        S - Return parameter set to zero or the number of the code
            segment.
        Hint - Desired SegHint.

function CurrentSegment: SegmentNumber;

    Abstract:

        CurrentSegment finds the segment number of its caller.

        Result: CurrentSegment = Segment number of the caller of
        CurrentSegment.
```

```
module MultiRead;

Written by Brad Myers    24 Jul 81.

Copyright (C) PERQ Systems Corporation, 1981, 1983.

    Abstract:

        This Module exports a procedure to read a file very quickly into
        memory.  The memory for the blocks of the file to be read must be
        allocated contiguously before the call is made.  Typically, this
        will be done by using CreateSegment.

Version Number V1.1
{/////////////////////////} EXPORTS {\\\\\\\\\\\\\\\\\\\\\\\\\}

Imports FileSystem from FileSystem;

Procedure MultiRead(fid: FileID; addr: pDirBlk; firstBlock,numBlocks:
integer);

Procedure MultiRead(fid: FileID; addr: pDirBlk; firstBlock,numBlocks:
    integer);.option 0

    Abstract:

        Does a multi-sector read on the file specified into the memory
        pointed to by addr

        NOTE: This only works for contiguous files.

    Parameters:
        fid - the fileID of the file to read from.
        addr - the address of the start of the memory to read the file
            into.  This must be pre-allocated.
        firstBlock - the logical block number of the first to read (the
            first legal value is 0; -1 will not work).
        numBlocks - the count of the number of blocks to transfer.
```

```
module PasLong;

PasLong - Extra stream package input conversion routines.
J. P. Strait & Michael R. Kristofic  ca. 15 Sep 81.
Copyright (C) PERQ Systems Corporation, 1981.

    Abstract:

        PasLong is the extra character input module of the Stream package.
        Its routines are called by code generated by the Pascal compiler
        in response to variations on Read, Readln, Write and Writeln
        statements. It is one level above Module Stream and uses Stream's
        lower-level input routines.

Version Number V2.2

exports

imports Stream from Stream;

procedure ReadD( var F: FileType; var X: long; B: integer );

    Abstract:

        Reads a double integer in free format with base B.  B can be any
        integer between 2 and 36, inclusive.

    Parameters:
        X - the double to be read.
        F - the file from which X is to be read.
        B - the base of X.  It may be any integer between 2 and 36,
            inclusive.  If B is less than zero and the user does not type
            an explicit plus or minus sign, X is read as an unsigned
            number.

    Errors:
        PastEof  -if an attempt is made to read F past the Eof.
        NotNumber - if non-numeric input is encountered in the file.
        LargeNumber - if the number is not in the range -2^31..2^32-1.
        BadBase - if the base is not in 2..36.

    Design: Number is read into the low order word of two double precision
        integers to avoid overflow.

procedure WriteD( var F: FileType; X: long; Field, B: integer );

    Abstract:

        Writes an double integer in fixed format with base B.

    Parameters:
        X - the double to be written.
        F - the file into which X is to be written.
        Field - the size of the field into which X is to be written.
        B - the base of X.  It is an integer whose absolute value must be
            between 2 and 36, inclusive.  If B is less than zero, X is
```

written as an unsigned number.

Errors:

BadBase -if the base is not in 2..36.

Design:

Value written from two double precision words avoids overflow.

```
module PasReal;

PasReal - Scott L. Brown     Created: 25-Nov-81
Copyright (C) 1981 - PERQ Systems Corporation
```

Abstract:

PasReal is an extra character input module of the Stream package. Its routines are called by code generated by the Pascal compiler in response to variations on Read, Readln, Write and Writeln statements. It is one level above Module Stream and uses Stream's lower-level input routines.

Version Number V0.2

```
exports

imports Stream from Stream;

procedure ReadR(var F: FileType;
                var value: real);

procedure WriteR(var F: FileType;
                     e: real;
                     TotalWidth: integer;
                     FracDigits: integer;
                     format: integer);

procedure ReadR(var F: FileType; var value: real);
```

Abstract:

This procedure reads a real number from the file F, and returns its value.

Parameters:

F - identifies the file from which to read.

value - is a return parameter returning the value of the real number read from the file F.

Results:

This procedure modifies the file buffer for F, and stores the value of the real number in value.

Side Effects:

This procedure reads characters from the external file F, until it receives a character which cannot be part of the real number, and it leaves this character in the file buffer.

There has been only minimal care taken with the file buffer, so if

an exception is raised during the read, there is no guarantee about its contents.

Exceptions:

PastEof - raised if an attempt is made to read beyond the end of file.

NotReal - raised if the stream of characters in file F does not correspond to a real number.

SmallReal - raised if the number read is too small to be represented by a 32-bit IEEE-Standard real number.

LargeReal - raised if the number read is too large to be represented by a 32-bit IEEE-Standard real number.

Calls:

StreamName - in module Stream for the stream name of file F.

GetC - in module Stream for reading the next character from file F.

RealMul

TenPower - returns a real number representing 10.0 raised to an integer (range -37..38) power.

Design:

The regular expression for real numbers is described by a DFA and implemented by a case statement, where each element of the case statement corresponds to a state in the DFA. This case statement stores information about the number read into (primarily) three variables: 1) mant - a string buffer containing the mantissa, 2) scale_factor - any adjustment to the mantissa (as a power of ten), and 3) exp - the exponent of the number read.

After the DFA has substringed out the mantissa characters, they are converted to a real number by successive divisions by 10.0.

To combine this information into a real number, the scale_factor is added to the exponent and this sum is the power of ten exponent of the mantissa. To combine sum with the mantissa, it has to be converted to a real number, which is done by the function TenPower. Then the result of TenPower is multiplied with the mantissa to produce the real number returned as value.

Much care has been taken to avoid calls to TenPower with actual parameters which are out of range, and also to minimize the number of real multiplications necessary (this to avoid error propogation).

```
procedure WriteR(var F: FileType; e: real; TotalWidth: integer; FracDigits:
    integer; format: integer);
```

Abstract:

This procedure writes a real number to a file F, under the given format specifications.

Parameters:

F - the file to which to write.

e - the value to write.

TotalWidth - the minimum number of characters to write.

FracDigits - the number of characters in the fractional part (used for fixed format only).

format - indicates either fixed or floating format.

Results:

The file buffer for F is modified by procedures nested in this one.

Calls:

StreamName - in module Stream for the stream name of file F.

base2_to_base10

normalize

Design:

There is some initialization, then if the real number to be written is zero, eWritten and ExpValue are obviously zero, else the real number needs to be converted to base 10 giving eWritten and ExpValue. Then a call is made to a procedure for the desired format.

The design here follows as much as possible the ISO Standard description for writing Pascal real numbers, in particular, the choice of many variable names.

```
module PERQ_String;

PERQ String hacking routines.
Written by: Donald Scelza
Copyright (C) 1980, 1981, 1982  PERQ Systems Corporation

     Abstract:

          This module implements the string hacking routines for the PERQ
          Systems PERQ Pascal.

Version Number V2.10
{********************} Exports  {********************}

Const MaxPStringSize=255;          { Length of strings}
Type PString = String[MaxPStringSize];

Procedure Adjust(Var STR: PString; LEN:Integer);
Function  Concat(Str1, Str2: PString): PString;
Function  Substr(Source: PString;  Index, Size: Integer): PString;
Procedure Delete(Var Str: PString;  Index, Size: Integer);
Procedure Insert(Var Source, Dest: PString; Index: Integer);
Function  Pos(Source, Mask: PString): Integer;

Function  PosC(s: PString; c: Char): Integer;
Procedure AppendString(var s1: PString; s2: PString);
Procedure AppendChar(var s: PString; c: Char);
Function  UpperCase(c: Char): Char;
Procedure ConvUpper(Var s: PString);

Exception StrBadParm;

     Abstract:

          Raised when bad index or length parameters passed to procedures or
          sometimes when string will be too long (other times, StrLong is
          raised in this case

          Function  RevPosC(s: PString; c: char): integer;
          Procedure PrependChar (c: char; var s: PString);
          Function  IntToStr (N: integer): PString;
          Function  Pad (PadCh: char; str: PString; len: integer): PString;
          Function  Upper (s: string): PString;

Procedure Adjust(var Str:PString; Len:Integer);

     Abstract:

          This procedure is used to change the dynamic length of a string.

     Parameters:

          Str is the string that is to have the length changed.

          Len is the new length of the string.  This parameter must be This
          value must be no greater than MaxPStringSize.
```

Environment: None

Results: This procedure does not return a value.

Side Effects: This procedure will change the dynamic length of Str.

Errors: If Len > MaxPStringSize then raise StrLong exception.

Design: Simple.

Function Concat(Str1,Str2:PString):PString;

Abstract:

This procedure is used to concatenate two string together.

Parameters:

Str1 and Str2 are the two strings that are to be concatenated.

Environment: None

Results: This function will return a single string as described by the parameters.

Errors: If Length(Str1) + Length(Str2) is greater then MaxPStringSize then raise StrLong exception.

Function SubStr(Source:PString; Index, Size:Integer):PString;

Abstract:

This procedure is used to return a sub portion of the string passed as a parameter.

Parameters:

Source is the string that we are to take a portion of.

Index is the starting position in Source of the substring.

Size is the size of the substring that we are to take.

Environment: None

Results: This function returns a substring as described by the parameter list.

Errors: If Index or Size are greater than MaxPStringSize then raise StrBadParm exception.

Procedure Delete(var Str:PString; Index, Size:Integer);

Abstract:

This procedure is used to remove characters from a string.

Parameters:

Str is the string that is to be changed. Characters will be removed from this string.

Index is the starting position for the delete.

Size is the number of character that are to be removed. Size characters will be removed from Str starting at Index.

Environment: None

Results: This procedure does not return a value.

Side Effects: This procedure will change Str.

Errors: None

Procedure Insert(var Source, Dest:PString; Index:Integer);

Abstract:

This procedure is used to insert a string into the middle of another string.

Parameters:

Source is the string that is to be inserted.

Dest is the string into which the inseration is to be made.

Index is the starting position, in Dest, for the inseration.

Environment: None

Results: This procedure does not return a value.

Side Effects: This procedure will insert Source in Dest starting at location Index.

Errors: If the resulting string is too long then generate a runtime error.

Procedure AppendString(var s1: PString; s2: PString);

Abstract:

puts s2 on the end of s1

Parameters : s1 is the left String and s2 goes on the end.

```
    Calls: PerqString.Concat.

    SideEffects : modifies s1.

Procedure AppendChar(var s: PString; c: Char);

    Abstract:

        puts c on the end of s

    Parameters : s is the left String and c goes on the end.

    SideEffects : modifies s.

Function UpperCase(c: Char): Char;

    Abstract:

        Changes c to uppercase if letter.

    Parameters:

        c is any char.

    Returns: char is uppercase if letter otherwise unchanged.

Procedure ConvUpper(Var s: PString);

    Abstract:

        Converts s to all upper case

    Parameters:

        s, passed by reference, to be converted

Function PosC(s: PString; c: char): integer;

    Abstract:

        Tests if c is a member of s.

    Parameters:

        c is any char; s is string to test for c member of.

    Returns: index of first c in s (from beginning of string) or zero if
        not there.
```

Function RevPosC(s: PString; c: char): integer;

Abstract:

Tests if c is a member of s.

Parameters:

c is any char; s is string to test for c member of.

Returns: index of first c in s (from end of string) or zero if not there.

Function Pos(Source, Mask:PString):Integer;

Abstract:

This procedure is used to find the position of a pattern in a given string.

Parameters:

Source is the string that is to be searched.

Mask is the pattern that we are looking for.

Environment: None

Results: If Mask occured in Source then the index into Source of the first character of Mask will be returned. If Mask was not found then return 0.

Side Effects: None

Errors: None

Procedure PrependChar (c: char; var s: PString);

Abstract:

puts c at the beginning of s

Parameters:

s is the original String and c goes on the front.

Side-Effects: modifies s if c is not Nul.

Calls: Adjust (from PERQ_String).

```
function IntToStr (N: integer): PString;
```

Abstract:

Converts the integer N to a string.

Calls: PrependChar.

```
function Pad (PadCh: char; str: PString; len: integer): PString;
```

Abstract:

Forces a string to a certain length using PadCh to left justify.

Returns: The longer string (preceeded by PadCh's).

Calls: PrependChar.

```
function Upper (s: string): PString;
```

Abstract:

Makes the ConvUpper procedure (of CmdParse) into a function.
Makes s into all capital letters.

Returns: the upper case equivalent of string s

Calls: ConvUpper from PERQ_String

```
module PMatch;

     Abstract:

          Does pattern matching on strings

          Patterns accepted are as follows:

          "*" matches 0 or more characters.

          "&" matches 1 or more characters.

          "#" matches exactly 1 character.

          "'0" matches any digit.

          "'A" matches any alphabetic (capitals only unless casefold).

          "'a" matches any alphabetic(lower case only unless 'casefold').

          "'@" matches any non-alphanumeric.

          "'*" matches '*', other patterns chars can be quoted also.

          Written by: Gene Ball at CMU

Version Number V2.6
{////////////////////////////} Exports {\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\}

Type pms255 = String[255];

Function PattMatch(var str,pattern: pms255; fold: boolean): boolean;
Function PattMap(var str,inpatt,outpatt,outstr:pms255; fold:boolean):
     boolean;
Procedure PattDebug(v: boolean);
Function IsPattern(var str: pms255): boolean;

Exception BadPatterns;

     Abstract:

          Raised if outPatt and inPatt do not have the same patterns in the
          same order for PattMap

Procedure PattDebug(v: boolean);

     Abstract:

          Sets the global debug flag

     Parameters:

          v is value to set debug to
```

```
    SideEffects: Changes debug value

Function IsPattern(var str: pms255): boolean;

    Abstract:

        Tests to see whether str contains any pattern matching characters

    Parameters:

        str - string to test.  If not pattern then removes all quotes.

    Returns: true if str contains any pattern matching characters; else
        false

Function PattMatch(var str, pattern: pms255; fold: boolean): boolean;

    Abstract:

        Compares str against pattern

    Parameters:
        str - full string to compare against pattern;
        pattern - pattern to compare against.  It can have special
            characters in it
        fold - determines whether upper and lower case are distinct.  If
            true then not.

    Returns: true string matches pattern; false otherwise

Function PattMap(var str,inpatt,outpatt,outstr:pms255;
                                    fold:boolean):boolean;

    Abstract:

        Compares str against inPatt, putting the parts of str that match
        inpatt into the corresponding places in outpatt and returning the
        result

        EXAMPLES:

        PattMap('test9.pas', 'test'0.pas', 'xtest'0.pas') => TRUE,
        'xtest9.pas'

        PattMap('test9.pas', '*.pas', '*.ada') => TRUE,  'test9.ada'

    Parameters:
        str - full string to compare against pattern;
        inpatt - pattern to compare against.  It can have special
            characters in it
        outpatt - pattern to put the parts of str into;  it must have the
            same special characters in the same order as in inpatt
        outStr - the resulting string if PattMap returns true;
        fold-determines whether upper and lower case are distinct.  It
            true then not.
```

Returns: true string matches pattern; false otherwise

Errors: Raises BadPatterns if outPatt and inPatt do not have the same patterns in the same order

```
module PopCmdParse;

    Abstract:

        This module provides procedures to help with PopUp menus.  See the
        module PopUp for the definition of pNameDesc and for some useful
        procedures for creating and destroying pNameDescs.

        Written by Brad Myers    Nov 18, 1981

        Copyright (C) 1981 PERQ Systems Corperation

Version Number V1.9
{******************} Exports {******************}

Imports CmdParse from CmdParse;
Imports PopUp from PopUp;

Function PopUniqueCmdIndex(Cmd: CString; Var names: pNameDesc): Integer;

Function GetCmdLine(Procedure IdleProc; prompt: String;
                    var line, cmd: CString; var inF: pCmdList;
                    var names: pNameDesc; var firstPress: boolean;
                    popOK: boolean): integer;

Function GetShellCmdLine(var cmd: CString; var inF: pCmdList;
                  var names: pNameDesc): integer;

Function GetConfirm(Procedure IdleProc; popOK: boolean;
                    prompt: String; def: integer;
                    var switches: pSwitchRec): integer;

Procedure NullIdleProc;

Procedure NullIdleProc;

    Abstract:

        This procedure does nothing.  It is useful as an IdleProc
        parameter to other procedures when no IdleProc is needed.

Function PopUniqueCmdIndex(Cmd: CString; Var names: pNameDesc): Integer;

    Abstract:

        This procedure is used to do a unique lookup in a popUp command
        table.  It is the same as UniqueCmdIndex except the table of names
        is the kind used by popUp menus.  If cmd is the full name of one
        of the names in names, even if is also a sub-part of other names,
        it is returned as the one found.

    Parameters:
        Cmd - the command that we are looking for.
        CmdTable - a table of the valid commands.  The first valid command
             in this table must start at index 1.
        NumCmds - the number of valid command in the table.
```

Returns: The index of Cmd in CmdTable. If Cmd was not found then return NumCmds + 1. If Cmd was not unique then return NumCmds+2.

```
Function GetShellCmdLine(var cmd: CString; var inF: pCmdList; var names:
    pNameDesc): integer;
```

Abstract:

This routine is similar to GetCmdLine except that it works on the command line specified to the Shell. It is should be used by programs that use GetCmdLine to parse the Shell command line. Command files are handled by GetShellCmdLine. The user can call ParseCmdArgs after GetShellCmdLine to get the arguments to the command.

Parameters:

cmd - the first command taken off the line. This will be valid even if the return value is greater than numCommands. It will be '' if no command found before the first significant break character.

inF - a command file list created by InitCmdFile. Just call InitCmdFile and pass in the inF returned. This procedure manages the list and handles all command files.

names - a variable length array of names used for popUp menus and for matching the input cmd against.

Returns:

Identical to GetCmdLine. Viz: index in the array or

numCommands + 1 ==> Name not found in array
numCommands + 2 ==> Name not unique
numCommands + 3 ==> Name was empty
numCommands + 4 ==> First command was a switch (it is in Cmd).
numCommands + 5 ==> Illegal character found after command.

```
Function GetCmdLine(* Procedure IdleProc; prompt: String;
                      var line, cmd: CString; var inF: pCmdList;
                      var names: pNameDesc; var firstPress: boolean;
                      popOK: boolean): integer *);
```

Abstract:

Reads a line from the input file. While waiting for a CR or a press do IdleProc. If press, then create a popUp window. Put name selected into line and return index. If type a line, put it into line. If first ID in line is not a switch then check to see if in names. If so, returns index. If not unique then returns numCommands+2. If not found then returns numCommands+1. If line was empty (naked CR or comment), then returns numCommands+3. numCommands+4 => switch. numCommands+5 => illegal character after command.

Parameters:

IdleProc - This procedure is called repeatedly until a full line is typed. It should execute quickly and not futz with the keyboard or stream. An application is a procedure that displays the time in the title line.

prompt - the prompt string to print for the user. Do not put the prompt separator (>) on the end of the prompt; GetCmdLine will do that for you. If reading from a command file, GetCmdLine change the prompt appropriately.

line - set to the line read including starting with the first significant character after the first command. It will not contain any comments.

cmd - the first command taken off the line. (It is not in line). This will be valid even if the return value is greater than numCommands. It will be '' if no command found before the first significant break character.

inF - a command file list created by InitCmdFile. Just call InitCmdFile and pass in the inF returned. This procedure manages the list and handles all command files. The application calling GetCmdLine will never see an '@'.

names - a variable length array of names used for popUp menus and for matching the input cmd against.

firstPress - USER MUST SET firstPress to true before first call to this procedure and then not modify it.

popOk - if true, then GetCmdLine will allow a PopUp menu when press on button to chose an item. If false, then no popUp allowed.

Returns:

index into the names array or:

numCommands + 1 ==> Name not found in array
numCommands + 2 ==> Name not unique
numCommands + 3 ==> Name was empty
numCommands + 4 ==> First command was a switch (it is in Cmd).
numCommands + 5 ==> Illegal character found after command.

Calls: NextIdString, PopUniqueCmdIndex, Menu (from PopUp), PopInit, IOSetModeTablet, IOCursorMode, FullLn, ReadLn, DestroyRes, DoCmdFile, ExitCmdFile, RemDelimiters

```
Function GetConfirm(* Procedure IdleProc; popOK: boolean;
                      prompt: String; def: integer;
                      var switches: pSwitchRec): integer *);
```

Abstract:

Handles a question that is answered Yes or No where the answer should come from the keyboard. Prompt followed by default (if any) is printed. Prompt may be null. If illegal input is typed, GetConfirm re-asks but doesn't use prompt.

Parameters:
IdleProc - This procedure is called repeatedly until a full line is typed. It should execute quickly and not futz with the keyboard or stream. An application is a procedure that displays the time in the title line.
prompt - the prompt to display for question.
default - index of the default answer: 1 = true or yes; 2 = false or no; other numbers mean no default.
popOK - tells whether a popUp window is allowed.
switches - set to NIL or a list of switches specified. Be sure to handle the switches first since one might be HELP.

Returns: 1 if true or yes.

2 if false or no.
3 if naked return when no default and switches <> NIL. This means that there was no argument but a switch was hit. If an answer is still needed, the application should re-call GetConfirm.

```
module PopUp;

Written by  Brad A. Myers 16-Nov-80

     Abstract:

          This program produces pop up windows that replace the screen area
          at a specified cursor location. The cursor is then changed and
          PopUp waits for a press.  Whenever the cursor is inside the
          window, the command at that point is highlighted.  If a press is
          done inside the window, the highlighted command is selected.  The
          user can control whether one or more than one command should be
          selected before window is removed.  If a press outside, no command
          is executed.  In any case, the window is erased and the original
          contents of that area is returned
```



```
Version Number V3.1

{-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-} EXPORTS {-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-}

Exception BadMenu;

     Abstract:

          Raised when parameters are illegal.

     Resume: NOT ALLOWED.

Exception Outside;

     Abstract:

          raised when press outside of menu.

     Resume: NOT ALLOWED.
Exception PopKeyHit;

     Abstract:

          raised when a key is hit and aborting on keys is enabled by
          calling AbortOnKey(true).  The key is left in the input buffer.

     Resume: NOT ALLOWED.

          Type s25 = String[25];
               NameAr = Array[1..1] of s25;
               pNameAr = ^NameAr;
               NameDesc = Record
                             header: s25;
                             numCommands: integer;
                             commands: NameAr;
                          End;
               pNameDesc = ^NameDesc;
```

```
        ResRes = ^ResArray;
        ResArray = Record
                    numIndices: integer;
                    pressVal: Integer; {TabMouse value when pressed}
                    indices: Array[1..1] of integer;
                   End;

    Procedure Menu(names: pNameDesc; isList: boolean;
                   first, last, curX, curY, maxYsize: integer; VAR
    res: ResRes);

    Procedure InitPopUp;
    Procedure DestroyRes(var res: ResRes);
    Procedure AbortOnKey(abort: boolean);
    Procedure AllocNameDesc(numNames, seg: Integer; Var names:
    pNameDesc);
    Procedure DestroyNameDesc(Var names: pNameDesc);

Procedure DestroyNameDesc(Var names: pNameDesc);
```

Abstract:

Delete names. It should have been created by AllocateNameDesc. The numCommands field better be the same as set when allocated.

Parameters:

names - The storage for names is deallocated and names is set to NIL.

```
Procedure InitPopUp;
```

Abstract:

creates cursors needed to make PopUp windows work. This should be called once before calling menu.

Environment: sets cursors. Sets the global abort on keyboard typing to false.

```
Procedure AbortOnKey(abort: boolean);
```

Abstract:

Sets the global flag that determines whether to abort on keyboard typing. If this is not called, then PopUp does not abort when there are keys. If this is called with TRUE, then the exception PopKeyHit is raised when a key is typed. The key is left in the input buffer.

```
Procedure DestroyRes(var res: ResRes);
```

Abstract:

Deallocates storage for res and sets it to NIL

Parameters:

res is ResRes to destroy

```
Procedure Menu(names: pNameDesc; isList: boolean;
    first, last, curX, curY, maxYsize: integer; VAR res: ResRes);
```

Abstract:

puts up a window with commands stacked vertically with the center at curX, curY. Allocates off the heap enough storage for old picture at that place so can restore it. Deallocates all storage when done.

Parameters:

names - a pointer to an array of names to put in the menu.

header - is put at the top of the menu. It may be empty in which case there is no header.

numcommands - the number of names in the array.

commands - an array of names to display. These can be generated by having pNameDesc with an array of the correct (or larger than the correct) size and recasting it into a pNameDesc, or by creating a segment to hold all the names that will be needed.

isList - if true says that a number of commands can be selected. if false, Menu returns as soon as the first command is selected.

first - the index of the first command in names^.commands to display. To display all items, use 1.

last - the index of the last command in names^.commands to display. To display all items, use names^.numCommands. Last must be greater than first.

curX - the x position at which to display the menu. If -1 then uses current pen position.

curY - the y position at which to display the menu. If -1 then uses current pen position.

maxYsize - the maximum size in bits of the menu. If -1, then menu will be big enough to hold all items (up to the size of the screen). maxYsize must be greater than 4*(fontHeight+4) which is 68 for the default font.

res - is set with an answer array. This array is allocated off the heap by Menu. Use DestroyRes to deallocate it. If Menu is exited via ^C or a press outside, then res is not allocated. The fields of res are set as follows:

numIndices - the number of items selected. If not isList then will be 1. If isList, will not be zero

indices - a variable length array of the indices of the names chosen. They are in increasing order irrespective of the order the names were picked.

Errors: Catches CtlC and raises CtlCAbort after removing the menu. Catches CtlShiftC and erases menu then re-raises CtlShiftC. Catches HelpKey and erases menu and then re-raises. If continued, then raises OutSide. Raises OutSide if press outside of the menu window. Raises BadMenu if parameters are illegal.

Environment: Requires enough memory be on the heap for picture. Requires that InitPopup has been called.

Procedure AllocNameDesc(numNames, seg: Integer; Var names: pNameDesc);

Abstract:

There are two ways to allocate the storage for a NameDesc. One is to declare in your program a type with an array of the correct size and the other fields exactly the same way. You then RECAST a pointer to that array into a pNameDesc. The other way is to use this procedure. It allocates the storage for numNames out of a segment. Turn off range checking when assigning or accessing the array.

NOTE: To deallocate the nameDesc returned, use DestroyNameDesc

Parameters:

numNames - the number of names in the array.

seg - the segment to allocate the nameDesc out of. If zero, then uses the default segment. This procedure uses NewP so the segment can have other things in it also.

names - set with the newly allocated pNameDesc. Its numCommands field is set with numNames. Do not change this size or the deallocation will not work.

```
module PopUpCurs;

Written by Brad A. Myers
Copyright (C) 1981 - PERQ Systems Corporation

Version Number V2.2
{-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-} EXPORTS
{-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-}

Type CursType = (Default, Select, Scroll, DoIt, Bar);
     FootAr = ARRAY [0..8] of ARRAY [0..3] of Integer;
     pFootAr = ^FootAr;

Procedure InitCurs;
Procedure DestroyCurs;
Procedure SetCurs(t: CursType);
Procedure InitFooter(VAR scrollP: pFootAr;  VAR spotP: pFootAr;
                         VAR footW: integer);

Procedure DestroyCurs;

     Abstract:

          Deallocates storage used for cursors

     SideEffects: Deallocates storage for cursors

     Environment: Must not be called before InitCursor is called

Procedure SetCurs(t: CursType);

     Abstract:

          Sets the cursor to the picture specified

     Parameters:

          t is the cursor picture to set to

Procedure InitCurs;

     Abstract:

          allocates storage used for cursors, and sets the cursors with the
          data for the pictures

     SideEffects: allocates storage for cursors
```

```
Procedure InitFooter(VAR scrollP: pFootAr;  VAR spotP: pFootAr;
    VAR footW: integer);
```

Abstract:

Allocates storage for pointers and fills in with right pictures.

Parameters:

all parameters are set by InitFooter. scrollp is the pointer to rasterOp from for the scroll bar and spotp is the pointer to the raster for the spot. footW is the width of the array.

```
module Profile;

    Abstract:

        This module is used to get information from the user profile file.

        Written by: Don Scelza

        Copyright (C) PERQ Systems Corperation, 1981

Version Number V1.1

(********************) Exports (********************)

This module provides facilities that will allow a program to get information
from the user profile.

The profile file is a text file that has the form:

    #<Subsystem name> <Line of text for that sub system>
    <More text for that subsystem>
    -
    -
    -
    #<Next subsystem>---

The base unit of the file is a text line.  The function that provides values
from the profile file will return a line of text each time that it is
called.  All text line between the #<Subsystem name> and the next
#<Subsystem name> are assumed to be assoicated with the first subsystem.
Successive calles to PFileEntry will return the next line of text for the
current subsystem.

Exception PNotFound(FileName: String);

    Abstract:

        Raised when profile file cannot be found

    Parameters:

        fileName is profile not found

Exception PNotInited;

    Abstract:

        Raised when a profile procedure is used but PFileInit not called
        first

Type ProfStr = String[255];

procedure PFileInit(PFileName, SubSystem: ProfStr);

function PFileEntry: ProfStr;
```

procedure PFileInit(PFileName, SubSystem: ProfStr);

Abstract:

This procedure is called each time a subsystem wishes to start to read information from the profile file. It is only called once per subsystem invocation. It will lookup the profile file and search for the required subsystem.

Parameters:

PFileName is the name of the profile file that is to be used.

SubSystem is the name of the subsystem that is to be searched for.

Side Effects: This procedure will change Inited, InLine and PFile.

Errors: If PFileName was not found then raise PNotFound.

function PFileEntry: ProfStr;

Abstract:

This procedure is used to get the next profile entry for a subsystem.

Results: This procedure will return the next line from the profile file for the current subsystem. If there are no more lines for the current subsystem return null.

Environment: PFileInit must have been called before this procedure is used. Uses the global InLine. Sets InLine to be empty.

Errors: If PFileInit was not called then raise PNotInited.

```
module QuickSort;

Copyright (C) 1981 PERQ Systems Corporation

Written by:  Mark G. Faust

     Abstract:

          Hoare's Quicksort algorithm with some simple optimizations.  This
          module provides two procedures, IntegerSort and StringSort, which
          sort arrays of integers and strings respectively.

          For a detailed description of the algorithm and references to
          papers on its analysis see [Robert Sedgewick, "Implementing
          Quicksort Programs," in CACM 21(10), 1978.]

          Because of rigid type checking of arrays in Pascal, pointers to
          the arrays to be sorted are passed along with an integer
          specifying the length of the array.  The procedures require that
          the array be declared [0..N+1] where the 0th through Nth elements
          are to be sorted.  The additional array element is used to speed
          up the sorting routine.  Before passing the array pointer to the
          sort procedure it is RECAST as either a IntegerArrayPtr or a
          StringArrayPtr.  An example for the integer sort is given below.
          The string sort is analogous.

          program ShowSort(input,output);

          imports QuickSort from QuickSort;

          const Size = 99;

          type MyArray = array[0..Size+1] of integer;

          var MyArrayPtr :^MyArray; i :integer;

          begin new(MyArrayPtr);

          for i := 0 to Size do readln(MyArrayPtr^[i]);

          IntegerSort(Size,recast(MyArrayPtr,PIntArray));

          for i := 0 to Size do writeln(MyArrayPtr^[i]); end.

exports

type

    ss25 = String[25];
    IntArray = array[0..0] of integer;
    StrArray = array[0..0] of ss25;

    PIntArray = ^IntArray;
    PStrArray = ^StrArray;
```

```
procedure IntegerSort(N :integer; A :PIntArray);
procedure StringSort(N :integer; A: PStrArray; Fold :boolean);
```

procedure IntegerSort(N :integer; A :PIntArray);

Abstract:

Given an integer N and a pointer to an array [0..N+1] of integers, sort [0..N] into ascending order using QuickSort.

Parameters:

N :integer One less than the upper bound of the array. It is the largest index of a valid key.

A :PIntArray A pointer to an array [0..N+1] of integers.

Results: The array from [0..N] is in ascending order

Side Effects: The array is sorted and the N+1st element contains MaxInt

procedure StringSort(N :integer; A :PStrArray; Fold :boolean);

Abstract:

Given an integer N and a pointer to an array [0..N+1] of strings, sort [0..N] into ascending lexigocraphic order using QuickSort. StringSort is case sensitive (e.g. A < a) unless Fold is True.

Parameters:

N :integer One less than the upper bound of the array. It is the largest index of a valid key.

A :PStrArray A pointer to an array [0..N+1] of strings.

Fold :boolean If True then we fold to UpCase for comparisons.

Results:

The array from [0..N] is in ascending order

Side Effects:

The array is sorted and the N+1st element contains the DEL character

```
module RandomNumbers;

J. P. Strait    15 Sep 80.
Copyright (C) PERQ Systems Corporation, 1980.

Module RandomNumbers contains two routines:

InitRandom - initializes the random number generator.

Random - a function which returns a new random number each time
it is referenced.

There is currently no way to seed the generator.

Random is a feedback shift-register pseudo-random number generator.
The algorithm used is one described in the article:

'Generalized Feedback Shift Register Pseudorandom Number
Generator'
T. G. Lewis and W. H. Payne
JACM Vol. 20, No. 3, July 1973, pp. 456-468.

Random produces multidimensional pseudo-random numbers equally
distributed in the interval -32768..32767 and has a period of
2^98.
Version Number V1.2
{////////////////////////////} exports {\\\\\\\\\\\\\\\\\\\\\\\\\\}

procedure InitRandom;
function Random: integer;

Procedure InitRandom;

    Abstract:

        Initialize the random number generator.  Every time this is
        called, the random numbers start over at the same place.

Function Random: integer;

    Abstract:

        returns a random 16-bit number
```

```
module Raster;

Copywrite (C) 1980 - The PERQ Systems Corporation

exports

  Const RRpl    = 0;        { Raster Op function codes }
        RNot    = 1;
        RAnd    = 2;
        RAndNot = 3;
        ROr     = 4;
        ROrNot  = 5;
        RXor    = 6;
        RXNor   = 7;

  Type RasterPtr = ^RasterArray; {a pointer that can be used as RasterOp
                                  or Line source and destination }
       RasterArray = Array[0..0] of integer;
```

```
module ReadDisk;

     Abstract:

          Module to Read and write to the disk using a buffer system

          Written by the CMU Spice Group

Version Number V1.5
{**********} exports {**********}

imports DiskIO from DiskIO;

function  ReadDisk(addr : DiskAddr)     : ptrDiskBuffer;
function  ChangeDisk(addr : DiskAddr)   : ptrDiskBuffer;
function  ReadHeader(addr : DiskAddr)   : ptrHeader;
function  ChangeHeader(addr : DiskAddr) : ptrHeader;
procedure FlushDisk(addr : DiskAddr);
procedure WriteDisk(addr : DiskAddr; ptr : ptrDiskBuffer; hdptr :
    ptrHeader);
procedure WriteHeader(addr : DiskAddr; ptr : ptrDiskBuffer; hdptr :
    ptrHeader);

procedure InitBuffers;
function  FindDiskBuffer(dskaddr : DiskAddr; alwaysfind : boolean) :
    integer;
procedure ReleaseBuffer(indx : integer);
procedure FlushBuffer(indx : integer);
procedure FlushAll;
procedure ChangeBuffer(indx : integer);
procedure ChgHdr(indx : integer);
procedure UseBuffer(indx,numtimes : integer);
function  BufferPointer(indx : integer) : ptrDiskBuffer;
function  HeaderPointer(indx : integer) : ptrHeader;
function  ReadAhead(addr : DiskAddr)    : ptrDiskBuffer;
procedure ForgetAll;

Exception FlushFail(msg: String; operation: DiskCommand; addr: DiskAddr;
     softStat: integer);

     Abstract:

          Raised when the system is unable to flush out a buffer.  The
          buffer is marked as flushed out, however, so the error will not
          repeat the next time a buffer needs to be flushed

     Parameters:

          Same as DiskFailure (in DiskIO)

     Resume: ALLOWED, but has no effect (procedure returns normally as if
          flush had been successful)
```

```
Function  ReadDisk(addr : DiskAddr) : ptrDiskBuffer;
```

Abstract:

Read the block specified and return the ptr of the buffer read into

Parameters:

addr is the address of the block to read

Returns: ptr to buffer read into

```
Function ReadAhead(addr : DiskAddr) : ptrDiskBuffer;
```

Abstract:

Identical to ReadDisk

Parameters:

addr is the address of the block to read

Returns: ptr to buffer read into

```
Function ReadHeader(addr : DiskAddr) : ptrHeader;
```

Abstract:

Reads block specified and returns a ptr to a buffer describing its header

Parameters:

addr is the address of the block to read

Returns: ptr to header read into

```
Function  ChangeDisk(addr : DiskAddr) : ptrDiskBuffer;
```

Abstract:

Reads block specified and returns a ptr to its data; in addition, mark file as changed so flush will write it out

Parameters:

addr is the address of the block to read

Returns: ptr to buffer holding the block read into

```
Function  ChangeHeader(addr : DiskAddr) : ptrHeader;
```

Abstract:

Reads block specified and returns a ptr to its data; in addition, mark file as header changed so flush will write it out using IOWriteFirst

Parameters:

addr is the address of the block to read

Returns: ptr to header read into

```
Procedure FlushDisk(addr : DiskAddr);
```

Abstract:

Removes block specified from buffer system and writes it out if changed. If addr not in buffer then NO-OP

Parameters:

addr is block to flush

```
Procedure WriteDisk(addr: DiskAddr; ptr: ptrDiskBuffer; hdptr: ptrHeader);
```

Abstract:

Writes out a block using DskWrite. If block for addr is in a buffer then Release it first.

Parameters:
addr - the address of the block to write
ptr - points to a buffer of data
hdptr - points to a buffer of header

```
Procedure WriteHeader(addr : DiskAddr; ptr : ptrDiskBuffer;
          hdptr : ptrHeader);
```

Abstract:

Writes out a block using DskFirstWrite. If block for addr is in a buffer then Release it first.

Parameters:

addr is the address of the block to write ptr points to a buffer of data hdptr points to a buffer of header

```
Procedure InitBuffers;

    Abstract:

        Initializes the buffer system

function FindDiskBuffer(dskaddr: DiskAddr; alwaysfind: boolean): integer;

    Abstract:

        Finds the buffer that contains the data for block dskAddr.

    Parameters:

        dskAddr  - is address to find buffer for alwaysFind tells whether
        to read in if not found

    Returns: Index of buffer found or zero if not there

Procedure ReleaseBuffer(indx : integer);

    Abstract:

        Mark the table entry as unused.

    Parameters:

        indx  - is entry to mark

Procedure FlushBuffer(indx : integer );

    Abstract:

        Write out the data for the buffer indx if changed and then mark
        the buffer as not changed.

    Parameters:

        indx  - is buffer to flush

    Errors: FlushFail raised if can't Flush buffer due to a write error

Procedure FlushAll;

    Abstract:

        Writes out the data for all the buffers and then mark them all as
        unchanged.

    Errors: FlushFail  - is raised if cannot Flush a buffer due to a write
        error. Does not stop at first error, but goes and tries all
        buffers before raising the exception
```

```
Procedure ChgHdr(indx : integer);

     Abstract:

          Mark a buffer as having its header changed.

     Parameters:

          indx  - is buffer to mark

Procedure UseBuffer(indx,numtimes : integer);

     Abstract:

          Mark a buffer as used.

     Parameters:
          indx  - is buffer to mark
          numTimes -  the number to incrememt use count by

Function  BufferPointer(indx : integer) : ptrDiskBuffer;

     Abstract:

          return the bufferPtr for a buffer.

     Parameters:

          indx  - is buffer

     Returns: Ptr to buffer

Function  HeaderPointer(indx : integer) : ptrHeader;

     Abstract:

          return the header Ptr for a buffer.

     Parameters:

          indx  - is buffer

     Returns: ptr to header
```

```
module Reader;

Reader - Stream package input conversion routines.
J. P. Strait   ca. 1 Jan 81.
Copyright (C) PERQ Systems Corporation, 1981.

     Abstract:

          Reader is the character input module of the Stream package.  It is
          called by code generated by the Pascal compiler in response to
          Read or Readln.  It is one level above Module Stream and uses
          Stream's lower-level input routines.

Version Number V2.1

exports

imports Stream from Stream;

procedure ReadBoolean( var F: FileType; var X: boolean );
procedure ReadCh( var F: FileType; var X: char; Field: integer );
procedure ReadChArray( var F: FileType; var X: ChArray; Max, Len: integer );
procedure ReadIdentifier( var F: FileType; var X: integer;
                          var IT: IdentTable; L: integer );
procedure ReadInteger( var F: FileType; var X: integer );
procedure ReadString( var F: FileType; var X: String; Max, Len: integer );
procedure ReadX( var F: FileType; var X: integer; B: integer );

procedure ReadBoolean( var F: FileType; var X: boolean );

     Abstract:

          Reads a boolean in free format.

     Parameters:
          X - the boolean to be read.
          F - the file from which X is read.

     Errors: PastEof  if an attempt is made to read F past the Eof.
          NotBoolean  if a non-boolean is encountered in the file.

procedure ReadCh( var F: FileType; var X: char; Field: integer );

     Abstract:

          Reads a character in fixed or free format.

     Parameters:
          X - the character to be read.
          F - the file from which X is to be read.
          Field - the size of the field X is in.
```

Errors: PastEof if an attempt is made to read F past the Eof.

```
procedure ReadChArray( var F: FileType; var X: ChArray; Max,
        Len: integer );
```

Abstract:

Reads a packed character array in free or fixed format. If free format reading is selected, spaces are skipped and characters are read until another space is encountered.

Parameters:
X - the character array to be read.
F - the file from which X is to be read.
Max - the declared length of X.
Len - the size of the field. Len <= 0 selects free format reading.

Errors: PastEof if an attempt is made to read F past the Eof.

```
procedure ReadIdentifier( var F: FileType; var X: integer;
        var IT: IdentTable; L: integer );
```

Abstract:

Reads an identifier and returns its position in a table. A table lookup is performed requiring only that the identifier typed uniquely matches the beginning of a single table entry.

Parameters:
X - set to the ordinal of the identifier read.
F - the file from which X is read.
IT - the table of identifiers indexed from 0 to L.
L - the largest identifier ordinal defined by the table.

Errors:
BadIdTable if length of the identifier table is less than 1.
PastEof if an attempt is made to read F past the Eof.
NotIdentifier if a non-identifier is encountered in the file.
IdNotDefined if the identifier is not in the table.
IdNotUnique if the identifier is not unique.

```
procedure ReadInteger( var F: FileType; var X: integer );
```

Abstract:

Reads a decimal integer in free format.

Parameters:
X - the integer to be read.
F - the file from which X is to be read.

Errors:
PastEof if an attempt is made to read F past the Eof.
NotNumber if non-numeric input is encountered in the file.
LargeNumber if the number is not in the range -32768..32767.

```
procedure ReadString( var F: FileType; var X: String; Max, Len: integer );
```

Abstract:

Reads a string in free or fixed format. If free format is selected, spaces are skipped and characters are read until another space is encountered.

Parameters:
X - the string to be read.
F - the file from which X is to be read.
Max - the declared maximum length of the string.
Len - the size of the field. Len <= 0 selects free format.

Errors: PastEof if an attempt is made to read F past the Eof.

```
procedure ReadX( var F: FileType; var X: integer; B: integer );
```

Abstract:

Reads an integer in free format with base B. B may be any integer between 2 and 36, inclusive.

Parameters:
X - the integer to be read.
F - the file from which X is to be read.
Field - the size of the field X is in.
B - the base of X. It may be any integer between 2 and 36, inclusive.

Errors:
PastEof if an attempt is made to read F past the Eof.
NotNumber if non-numeric input is encountered in the file.
LargeNumber if the number is not in the range -32768..32767.
BadBase if the base is not in 2..36.

```
module RealFunctions;

RealFunctions - Standard functions for reals.
J. Strait  27 Nov 81.
Copyright (C) PERQ Systems Corporation, 1981.
```

Abstract:

RealFunctions implements many of the standard functions whose domain and/or range is the set of real numbers. The implementation of these functions was guided by the book

Software Manual for the Elementary functions, William J. Cody, Jr. and William Waite, (C) 1980 by Prentice-Hall, Inc.

The domain (inputs) and range (outputs) of the functions are given in their abstract. The following notation is used. Parentheses () are used for open intervals (those that do not include the endpoints), and brackets [] are used for closed intervals (those that do include their endpoints). The closed interval [RealMLargest, RealPLargest] is used to mean all real numbers, and the closed interval [-32768, 32767] is used to mean all integer numbers.

DISCLAIMER:

Only the most cursory testing of these functions has been done. No guarantees are made as to the accuracy or correctness of the functions. Validation of the functions must be done, but at some later date.

Design: AdX, IntXp, SetXp, and Reduce are implemented as Pascal functions. It is clear that replacing the calls with in-line code (perhaps through a macro expansion) would improve the efficiency.

Many temporary variables are used. Elimination of unnecessary temporaries would also improve the efficiency.

Many limit constants have been chosen conservatively, thus trading a small loss in range for a guarantee of correctness. The choice of these limits should be re-evaluated by someone with a better understanding of the issues.

Some constants are expressed in decimal (thus losing the guarantee of precision). Others are expressed as Sign, Exponent, and Significand and are formed at execution time. Converting these two 32-bit constants which are Recast into real numbers would improve the correctness and efficiency.

More thought needs to be given to the values which are returned after resuming from an exception. The values that are returned now are the ones recommended by Cody and Waite. It seems that Indefinite values (NaNs in the IEEE terminology) might make more sense in some cases.

```
Version Number V1.5
```

```
exports

const RealPInfinity =   Recast(#17740000000,Real);   {  1.0 / 0.0 }
      RealMInfinity =   Recast(#37740000000,Real);   { -1.0 / 0.0 }
      RealPIndefinite = Recast(#00000000001,Real);   {  0.0 / 0.0 }
      RealMIndefinite = Recast(#20000000001,Real);   { -0.0 / 0.0 }
      RealPLargest =    Recast(#17737777777,Real);   { largest positive }
      RealMLargest =    Recast(#37737777777,Real);   { largest negative }
      RealPSmallest =   Recast(#00040000000,Real);   { smallest positive }
      RealMSmallest =   Recast(#20040000000,Real);   { smallest negative }

function Sqrt( X: Real ): Real;
function Exp( X: Real ): Real;
function Ln( X: Real ): Real;
function Log10( X: Real ): Real;
function Power( X, Y: Real ): Real;
function PowerI( X: Real; Y: Integer ): Real;
function Sin( X: Real ): Real;
function Cos( X: Real ): Real;
function Tan( X: Real ): Real;
function CoTan( X: Real ): Real;
function ArcSin( X: Real ): Real;
function ArcCos( X: Real ): Real;
function ArcTan( X: Real ): Real;
function ArcTan2( Y, X: Real ): Real;
function SinH( x:real ) : real;
function CosH( x:real ) : real;
function TanH( x:real ) : real;

exception SqrtNeg( X: Real );
```

Abstract:

SqrtNeg is raised when Sqrt is passed a negative argument. You may resume from this exception, in which case Sqrt returns Sqrt(Abs(X)).

Parameters:

X - Argument of Sqrt.

```
exception ExpLarge( X: Real );
```

Abstract:

ExpLarge is raised when Exp is passed an argument which is too large. You may resume from this exception, in which case Exp returns RealPInfinity.

Parameters:

X - Argument of Exp.

```
exception ExpSmall( X: Real );
```

Abstract:

ExpLarge is raised when Exp is passed an argument which is too small. You may resume from this exception, in which case Exp returns 0.0.

Parameters:

X - Argument of Exp.

```
exception LogSmall( X: Real );
```

Abstract:

LogSmall is raise when Ln or Log10 is passed an argument which is too small. You may resume from this exception in which case Ln or Log10 returns RealMInfinity if X is zero or the log of Abs(X) if X is non-zero.

Parameters:

X - Argument of Ln or Log10.

```
exception PowerZero( X, Y: Real );
```

Abstract:

PowerZero is raised when Power or PowerI is called with X = 0.0 and Y = 0.0. You may resume from this exception in which case Power or PowerI returns RealPInfinity.

Parameters:
X - Argument of Power or PowerI.
Y - Argument of Power or PowerI.

```
exception PowerNeg( X, Y: Real );
```

Abstract:

PowerNeg is raised when Power is called with X < 0.0 or with X = 0.0 and Y < 0.0, or PowerI is called with X = 0.0 and Y < 0. You may resume from this exception in which case Power or PowerI returns Power(Abs(X),Y) in the case of X < 0.0 or returns RealPInfinity in the case of X = 0.0 and Y < 0.0.

Parameters:
X - Argument of Power or PowerI.
Y - Argument of Power or PowerI.

```
exception PowerBig( X, Y: Real );
```

Abstract:

PowerBig is raised when Power or PowerI is called with X and Y for which X raised to the Y power is too large to be represented. You may resume from this exception in which case Power or PowerI returns RealPInfinity.

Parameters:
X - Argument of Power or PowerI.
Y - Argument of Power or PowerI.

```
exception PowerSmall( X, Y: Real );
```

Abstract:

PowerSmall is raised when Power or PowerI is called with X and Y for which X raised to the Y is too close to zero to be represented. You may resume from this exception in which case Power or PowerI returns 0.0.

Parameters:
X - Argument of Power or PowerI.
Y - Argument of Power or PowerI.

```
exception SinLarge( X: Real );
```

Abstract:

SinLarge is raised when Sin is called with an argument which is too large. You may resume from this exception in which case Sin returns 0.0.

Parameters:

X - Argument of Sin.

```
exception CosLarge( X: Real );
```

Abstract:

CosLarge is raised when Cos is called with an argument which is too large. You may resume from this exception in which case Cos returns 0.0.

Parameters:

X - Argument of Cos.

```
exception TanLarge( X: Real );
```

Abstract:

CosLarge is raised when Tan or CoTan is called with an argument which is too large. You may resume from this exception in which case Tan or CoTan returns 0.0.

Parameters:

X - Argument of Tan or CoTan.

```
exception ArcSinLarge( X: Real );
```

Abstract:

ArcSinLarge is raised when ArcSin is called with an argument which is too large. You may resume from this exception in which case ArcSin returns RealPInfinity.

Parameters:

X - Argument of ArcSin.

```
exception ArcCosLarge( X: Real );
```

Abstract:

ArcCosLarge is raised when ArcCos is called with an argument which is too large. You may resume from this exception in which case ArcCos returns RealPInfinity.

Parameters:

X - Argument of ArcCos.

```
exception ArcTan2Zero( Y, X: Real );
```

Abstract:

ArcTan2Zero is raised when ArcTan2 is called with both X and Y equal to zero. You may resume from this exception in which case ArcTan2 returns RealPInfinity.

Parameters:
Y - Argument of ArcTan2.
X - Argument of ArcTan2.

```
exception SinHLarge( X: Real );
```

Abstract:

SinHLarge is raised when the arqument to SinH would cause a result whose magnitude is too large to be represented on the Perq. Note that SinH is implemented (for now at least) is terms of the Exp function and that function is the bound on SinH domain.

Parameters:

X - Argument of SinH

```
exception CosHLarge( X: Real );
```

Abstract:

CosHLarge is raised when the arqument to CosH would cause a result whose magnitude is too large to be represented on the Perq. Note that CosH is implemented (for now at least) is terms of the Exp function and that function is the bound on CosH domain.

Parameters:

X - Argument of CosH

```
function Sqrt( X: Real ): Real;
```

Abstract:

Compute the square-root of a number.

Domain = [0.0, RealPLargest]. Range = [0.0, Sqrt(RealPLargest)].

Parameters:

X - Input value.

Returns: Square-root of X.

```
function Ln( X: Real ): Real;

    Abstract:

        Compute the natural log of a number.

        Domain = [0.0, RealPLargest].
        Range  = [RealMLargest, Ln(RealPLargest)].

    Parameters:

        X - Input value.

    Returns: Natural log of X.

function Log10( X: Real ): Real;

    Abstract:

        Compute the log to the base 10 of a number.

        Domain = [0.0, RealPLargest].
        Range  = [RealMLargest, Log10(RealPLargest)].

    Parameters:

        X - Input value.

    Returns: Log to the base 10 of X.

    Calls: Ln

function Exp( X: Real ): Real;

    Abstract:

        Compute the exponential function.

        Domain = [-87.336, 88.722].
        Range  = (0.0, RealPLargest].

    Parameters:

        X - Input value.

    Returns:  e raised to the X power.
```

```
function Power( X, Y: Real ): Real;

    Abstract:

        Compute the result of an arbitrary number raised to an arbitrary
        power.

        DomainX = [0.0, RealPLargest].
        DomainY = [RealMLargest,RealPLargest].
        Range   = [0.0, RealPLargest].

        With the restrictions that

        1) if X is zero, Y must be greater than zero.

        2) X raised to the Y is a representable real number.

    Parameters:
        X - Input value.
        Y - Input value.

    Returns:  X raised to the Y power.

function PowerI( X: Real; Y: Integer ): Real;

    Abstract:


        Compute the result of an arbitrary number raised to an arbitrary
        integer power.  The difference between Power and PowerI is that
        negative values of X may be passed to PowerI.

        DomainX = [RealMLargest, RealPLargest]. DomainY = [-32768, 32767].
        Range   = [RealMLargest, RealPLargest].

        With the restrictions that
        1) if X is zero, Y must be non-zero.
        2) X raised to the Y is a representable real number.

    Parameters:
        X - Input value.
        Y - Input value.


    Returns:  X raised to the Y power.
```

```
function Sin( X: Real ): Real;

     Abstract:

          Compute the sin of a number.

          Domain = [-12867, 12867].
          Range =  [-1.0, 1.0].

     Parameters:

           X - Input value.

     Returns: Sin of X.

function Cos( X: Real ): Real;

     Abstract:

          Compute the cosin of a number.

          Domain = [-12867, 12867].
          Range =  [-1.0, 1.0].

     Parameters:

           X - Input value.

     Returns: Cos of X.

function Tan( X: Real ): Real;

     Abstract:

          Compute the tangent of a number.

          Domain = [-6433.0, 6433.0].
          Range  = [RealMInfinity, RealPInfinity].

     Parameters:

           X - Input value.

     Returns: Tangent of X.
```

```
function CoTan( X: Real ): Real;

     Abstract:

          Compute the cotangent of a number.

          Domain = [-6433.0, 6433.0].
          Range  = [RealMInfinity, RealPInfinity].

     Parameters:

           X - Input value.

     Returns: Cotangent of X.

function ArcSin( X: Real ): Real;

     Abstract:

          Compute the arcsin of a number.

          Domain = [-1.0, 1.0).
          Range  = [-Pi/2, Pi/2).

     Parameters:

           X - Input value.

     Returns: Arcsin of X.

     Design: It seems that the Domain and Range ought to be closed
          intervals, however this implementation apparently returns a number
          very close to zero when X is 1.0, rather than returning Pi/2 as it
          should.

function ArcCos( X: Real ): Real;

     Abstract:

          Compute the arccosin of a number.

          Domain = (-1.0, 1.0].
          Range  = (-Pi/2, Pi/2].

     Parameters:

           X - Input value.
```

Returns: Arccosin of X.

Design: It seems that the Domain and Range ought to be closed intervals, however this implementation apparently returns a number very close to zero when X is -1.0, rather than returning -Pi/2 as it should.

```
function ArcTan( X: Real ): Real;
```

Abstract:

Compute the arctangent of a number.

Domain = [RealMLargest, RealPLargest].
Range = (-Pi/2, Pi/2).

Parameters:

X - Input value.

Returns: Arctangent of X.

Design: Seems fine except for very large numbers.

```
function ArcTan2( Y, X: Real ): Real;
```

Abstract:

Compute the arctangent of the quotient of two numbers. One interpretation is that the parameters represent the cartesian coordinate (X,Y) and ArcTan2(Y,X) is the angle formed by (X,Y), (0,0), and (1,0).

DomainY = [RealMLargest, RealPLargest].
DomainX = [RealMLargest, RealPLargest].
Range = [-Pi, Pi].

Parameters:
Y - Input value.
X - Input value.

Returns: Arctangent of Y / X.

Design: Seems fine except for very large Y/X.

```
function SinH( x:real ) : real;
```

Abstract:

Compute the Hyperbolic Sine of a number.

Domain = [-87.33,87.33].
Range = [RealMLargest, RealPLargest].

Parameters:

X - Input value.

Returns: Hyperbolic Sine of X.

```
function CosH( x:real ) : real;
```

Abstract:

Compute the Hyperbolic Cosine of a number.

Domain = [-87.33,87.33].
Range = [1.0, RealPLargest].

Parameters:

X - Input value.

Returns: Hyperbolic Cosine of X.

```
function TanH( x:real ) : real;
```

Abstract:

Compute the Hyperbolic Tangent of a number.

Domain = [-8.66433975625,8.664339756251].
Range = [-1.0, 1.0].

Parameters:

X - Input value.

Returns: Hyperbolic Tangent of X.

```
module RS232Baud;
    Abstract:

        RS232Baud - set RS232 baud rate with optional input enable.

        J. P. Strait        21 Aug 80.

        Copyright (c) PERQ Systems Corporation 1980, 1981, 1982, 1983.
Version Number V1.2
exports
procedure SetBaud(Baud: String; Enable: Boolean);
Exception BadBaudRate;
    Abstract:

        Raised if Baud is not a valid baud rate.

        procedure SetRS232Port(Baud: string; Device: Integer);
Exception BadRSDevice;
    Abstract:

        Raised if Device is not an RS232 port.
procedure SetBaud(Baud: String; Enable: Boolean);
    Abstract:

        Sets the baud rate to baud specified by string arg

    Arguments:
        Baud - string of new baud rate (e.g. "2400")
        Enable - is ignored

    SideEffects: Changes status of RS232

    Errors: Raises BadBaudRate if string is illegal
procedure SetRS232Port(Baud: string; Device: Integer);
    Abstract:

        Sets the baud rate to baud specified by string arg

    Arguments:
        Baud - string of new baud rate (e.g. "2400")
        Device - chooses RSA or RSB
```

SideEffects: Changes status of RS232

Errors: Raises BadBaudRate if string is illegal

```
module RunRead;

RunRead - Module to read run files.
John P Strait   9 Apr 81.
CopyRight (C) PERQ Systems Corporation, 1981.

     Abstract:

          RunRead exports procedures to read and write run files.

          Design:

          If and when the format of run files is changed, the constant
          RFileFormat in module Code must be changed.  This is necessary so
          that the procedures to read run files will not crap out.

Version Number V1.2

exports

const RunReadVersion = '1.2';

imports Code from Code;

procedure ReadRunFile( var RunFile: RunFileType; Seg: Integer;
                       var Header: RunInfo;
                       var FirstSeg, FirstUserSeg, LastSeg: pSegNode;
                       ImportsWanted: Boolean );

procedure ReadSegNames( var RunFile: RunFileType; Seg: Integer;
                        FirstUserSeg: pSegNode );

procedure ReadRunFile( var RunFile: RunFileType; Seg: Integer;
     var Header: RunInfo; var FirstSeg, FirstUserSeg, LastSeg: pSegNode;
     ImportsWanted: Boolean );

     Abstract:

          ReadRunFile reads a run file and builds a structure that
          represents that run file.  The run file is read up to, but not
          including, the names of the .Seg files.

     Parameters:
          RunFile  - A file variable which has been Reset to the desired
               file. ReadRunFile does *not* close the file.
          Seg      - Segment number for dynamic allocation.
          Header   - The RunInfo record.
          FirstSeg - Set to point to the first segment in the run file.
          FirstUserSeg - Set to point to the first user segment in the run
               file.
          LastSeg  - Set to point to the last segment in the run file.
          ImportsWanted - True iff Import entries are to be read from the
               run file.
```

```
procedure ReadSegNames( var RunFile: RunFileType; Seg: Integer;
        FirstUserSeg: pSegNode );
```

Abstract:

ReadSegNames reads .Seg file names from a run file and adds them to a structure that represents that run file.

Parameters:

RunFile - A file variable which has been Reset to the desired file and already read with ReadRunFile. ReadSegNames does *not* close the file.

Seg - Segment number for dynamic allocation.

FirstUserSeg - A pointer to the first user segment in the run file.

```
module RunWrite;

RunWrite - Module to write run files.
John P Strait   9 Apr 81.
CopyRight (C) PERQ Systems Corporation, 1981.

     Abstract:

          RunWrite exports procedures to write run files.

          Design: If and when the format of run files is changed, the
          constant RFileFormat in module Code must be changed.  This is
          necessary so that the procedures to read run files will not crap
          out.

Version Number V1.2

exports

const RunWriteVersion = '1.2';

imports Code from Code;

procedure WriteRunFile( var RunFile: RunFileType; Header: RunInfo;
                        FirstSeg, FirstUserSeg: pSegNode );

procedure WriteRunFile( var RunFile: RunFileType; Header: RunInfo;
          FirstSeg, FirstUserSeg: pSegNode );

     Abstract:

          ReadRunFile writes a run file from a structure that represents
          that run file.

     Parameters:
          RunFile  - A file variable which has been Rewritten to the desired
               file.  WriteRunFile does *not* close the file.
          Header   - The RunInfo record.
          FirstSeg - A pointer to the first segment in the run file.
          FirstUserSeg - A pointer to the first user segment in the run
               file.
```

```
module Screen;

Written By: Miles A. Barel     July 1, 1980
PERQ Systems Corporation
Pittsburgh, PA 15213

     Abstract:

          Provides the interface to the PERQ screen including rudimentary
        support for multiple windows

exports

Imports Raster from Raster;

Version Number V4.4

Const   ScreenVersion = 'V4.3';
        VarWin = false;  {if true then can have an arbitrary number of
                          windows and storage for them has to be
                          allocated off a heap.  If false then there
                          are 17 windows max, and storage is in
                          screens global data.
                          NOTE: There are still bugs in VarWin true}

Type
        FontPtr = ^Font;
        Font = Packed Record    { Contains character sets }
                Height:integer; { Height of the KSet }
                Base: integer;  { distance from top of characters to base-
                                  line }
                Index: Array [0..#177] of { Index into character patterns }
                    Packed Record case boolean of
                      true: (Offset: 0..767; { position of character in
                                               patterns
                             Line: 0..63;    { Line of patterns containing char }
                             Width: integer); { Width of the character }
                            false:(Loc:integer; Widd: integer)
                    end;
                Filler: array[0..1] of integer;
                Pat: Array [0..0] of integer;  { patterns go here }
                                               { We turn off range checking to }
                                               { access patterns, hence allowing }
                                               { KSets of different sizes }
            end;
{$ifc VarWin then}
     WindowP = ^WindowType;
{$endc}
     WindowType = Packed Record
{$ifc VarWin then}
                     winNumber: Integer;  {this window number}
{$endc}
                     winBY, winTY, winLX, winRX,   { Limits of window
                                                     area }
```

```
                        winHX, winHY, winMX, winMY,   { Limits of useable
                                                        area }
                        winCurX, winCurY, winFunc: integer;
                        winKSet: FontPtr;
                        winCrsChr: char;
                        winHasTitle, winCursorOn, defined: boolean;
{$ifc VarWin then}
                        winNext: WindowP;
{$endc}
                        end;
{$ifc VarWin then}
Const   MaxWIndx = 32767;
{$elsec}
Const   MaxWIndx = 32;
{$endc}

Type    WinRange = 0..MaxWIndx;
        LineStyle = (DrawLine,EraseLine,XorLine);

        LS = String[255];

Const PortraitWordWidth = 48;
      PortraitBitWidth = 768;
      PortraitBitHeight = 1024;

Const LandscapeWordWidth = 80;
      LandscapeBitWidth = 1280;
      LandscapeBitHeight = 1024;

      TitStrLength = 255; {big so can fit lots of characters across a
                           landscape screen.  The actual number allowed now
                           is in the variable SNumTitleChars}


      KSetSLen = 48;          { Scan Line Length used by Fonts }

Type  STitStrType = String[TitStrLength];

Var   SScreenW: Integer;   {word width of screen; use when want Screen
                            in RasterOp or Line}
      SScreenP: RasterPtr; {for use when want Screen in RasterOp or Line}
      SBitWidth: Integer;  {bit width of current screen}
      SBitHeight: Integer; {bit height of current screen}
      SMaxBitHeight: Integer;  { maximum possible bit height for this screen
                                 type; <> SBitHeight if screen shrunk at
                                 start up }
      SIsLandScape: boolean; {true if landscape, else false}
      SNumTitleChars: Integer;  {number of characters that will fit in a
                                 full width title line USING THE STANDARD
                                 FONT}
      SCursorOn: boolean;
      SFunc: integer;       { Raster-op function for SPutChr }

      SCurBitHeight: Integer;  { current BitHeight.  Will always be
                                 <= SBitHeight;  will be < if current screen
                                 shrunk }
```

```
{$ifc VarWin then}
      FirstWindp,              { first window's pointer; better not be NIL }
      CurWindp: WindowP;       { current window's pointer }
{$elsec}
      CurWind: WinRange;
      WinTable: Array[WinRange] of WindowType;
{$endc}

Procedure ScreenInit;            { CALL THIS ONCE AT BOOT }
Procedure ScreenReset;           { This procedure de-allocates storage for
                                   all windows and sets up the default
window.}
Procedure SPutChr(CH:char);      { put character CH out to current position }
                                 { on the screen.  Chars FF, CR, and LF }
                                 { have special meanings unless #200 bit
set:}
                                 {       FF - clear screen
                                 {       CR - move left to margine
                                 {       LF - move vertically down one
                                 {       BS - erase previous character   }
Procedure SSetCursor(X,Y: integer);     { Set Cursor Position to X,Y }
Procedure SReadCursor(var X,Y: integer);{ Read Cursor Position }
Procedure SCurOn;                       { Enable display of Cursor }
Procedure SCurOff;                      { Disable display of Cursor }
Procedure SCurChr(C: char);             { Set cursor character }
Procedure SChrFunc(F: integer);         { Set raster-op function for
SPutChr}
Procedure SSetSize(Lines: integer; complemented, screenOff: Boolean);
                                        { Set Screen Size; lines must be a
                                          multiple of 128; screenOff if true
                                          turns off display in part below
                                          lines in which case, complemented
                                          describes off part of screen }
Procedure CreateWindow(WIndx: WinRange;
                  OrgX, OrgY, Width, Height: integer; Title: STitStrType);
Procedure ChangeWindow(WIndx: WinRange);
Procedure GetWindowParms(var WIndx: WinRange;
          var OrgX, OrgY, Width, Height: integer; var hasTitle: Boolean);
Procedure ChangeTitle(Title: STitStrType);
Procedure SetFont(NewFont: FontPtr);
Function GetFont: FontPtr;

Procedure SClearChar(c: Char; funct: Integer); {delete prev char}
                                        { c BETTER NOT be CR or LF}
Procedure Line(Style: LineStyle; X1, Y1, X2, Y2: integer; Origin:
    RasterPtr);
Procedure SVarLine(Style: LineStyle; X1, Y1, X2, Y2, Width: integer;
                  Origin: RasterPtr);
Procedure SBackSpace(c: Char); {move back over last char of curLine}
                                        { c BETTER NOT be CR or LF}
Procedure RefreshWindow(WIndx: WinRange); {redraws window outline and title
                                           area. DOES NOT REDRAW TITLE}
Procedure StartLine;
Procedure ToggleCursor;
Procedure NewLine;
```

```
Procedure SaveLineEnd(x: Integer);
Procedure SFullWindow;

Exception WBadSize; {parameter to SSetSize bad}
```

Abstract:

Raised if the lines parameter to SSetSize is not a multiple of 128 or is <=0. Also raised if a window is totally below area to release so will disappear then if window # 0 or is the current window, then Raises WBadSize.

```
Exception BadWNum;  {indx is invalid}
```

Abstract:

Raised if a window number parameter is illegal (not defined or out of range.

```
Exception WTooBig;
```

Abstract:

Raised if parameters for new window specify an area that would extend off screen.

```
Exception CursOutSide;
```

Abstract:

Raised if try to set the cursor outside of the current window.

Resume: Allowed. If resume, then cursor is NOT moved (same effect as if signal is caught but not resumed).

```
Procedure StartLine;
```

Abstract:

Resets Curline and variables describing the current line start.

```
Procedure ToggleCursor;
```

Abstract:

Inverts Cursor picture.

SideEffects: Changes the picture on the screen;

```
Procedure SSetCursor(x,y: integer);

    Abstract:

        Moves the cursor to the specified screen position.

    Parameters:
        x and y are Screen position where the next char will go.  Note
            that y specified the BOTTOM of the character.

    SideEffects: Changes the cur char positions AND sets line to be empty
        (so BS won't work);

    Errors: Raises CursOutside if try to set the cursor outside the current
        window

Procedure SReadCursor(var x,y:integer);

    Abstract:

        Returns the current screen coords for chars.

    Parameters:

        x and y are set to the Screen position where the next char will go

Procedure SCurOn;

    Abstract:

        Turns the char cursor on.

    SideEffects: Changes SCursorOn global vble

Procedure SCurOff;

    Abstract:

        Turns the char cursor off.

    SideEffects: Changes SCursorOn global vble

Procedure SCurChr(C: char);

    Abstract:

        Set the character to be used as the cursor.

    SideEffects: Changes the cursor character
```

```
Procedure SChrFunc(F: integer);
```

Abstract:

Set the function to be used for drawing chars to the screen.

SideEffects: Changes the char function

```
Procedure SSetSize(Lines: integer; complemented, screenOff: Boolean);
```

Abstract:

Change the size of the screen so rest of memory can be used for other things (if smaller)

Parameters:

Lines is the number of lines in the displayed part of the screen. It must be a multiple of 128 and > 0. Complemented describes the off part of the screen and screenOff determines whether it is displayed (false) or not; if displayed then complemented determines whether it is erased white or black.

Errors: if lines a bad value then Raises WBadSize. If a window is totally below area to release and will disappear then if window # 0 or is the current window, then Raises WBadSize.

SideEffects: Changes the values describing windows. If a window is totally below area to release and will disappear then if not window # 0 or is the current window, then makes the window undefined.

```
Procedure NewLine;
```

Abstract:

Moves the cursor to the next line scrolling if necessary; DOES NOT do a CR

SideEffects: Changes the cursor position and may scroll

```
Procedure SaveLineEnd(x: Integer);
```

Abstract:

Saves x as the end of a line

Parameters:

x is the xPos of the end of a line

SideEffects: puts x at the end of LineEnds table; increments lastLineEnd; if table is full then scrolls table

```
Procedure SBackSpace(c: Char);

    Abstract:

        Move the cursor back over c; c BETTER NOT be CR or LF

    Parameters:

        c is the character to backspace over.

    SideEffects: Moves the cursor back the width of char c; (DOES NOT ERASE
        CHAR)

Procedure SClearChar(c: char; funct: Integer);

    Abstract:

        Deletes the c from screen; c BETTER NOT be CR or LF

    Parameters:

        c is char to be erased; funct is RasterOp function to use in
        deleting char.  It should be RXor if chars are black on white and
        RXNor if chars are white on black.

    SideEffects: erases the last char of line;

Procedure SPutChr(CH: Char);

    Abstract:

        Write a char into the current window

    Parameters:

        Ch is char to write.  If #200 bit is not set, checks to see if
        char is one of Bell, BS, FF, LF, CR and does something special.

    SideEffects: Writes char to screen, moves cursor; may do a NewLine (and
        scroll) if at end of Line

Procedure ChangeTitle(Title: STitStrType);

    Abstract:

        Changes the title of the current window (and displays new one).

    Parameters:

        Title is new string.  Characters in it are quoted so special
        characters will be displayed.
```

SideEffects: Changes title on screen

Procedure CreateWindow(WIndx: WinRange; OrgX,OrgY,Width,Height: integer; Title:STitStrType);

Abstract:

Creates new window for Windx (or overwrites old values for that window) and makes it the current window. Writes title (IN CURRENT FONT) if title <> '';

Parameters:

WIndx is index to use for the window created; OrgX and OrgY are the upper left corner of the outside of the new window (chars will be at least 5 bits in from that). Width and Height are total outside values for window (NOT the width and height of the character area). Title is title for window. If not '' then hairlines and a black area are put around window.

SideEffects: Writes current values into current window; creates a new window and erases its area on screen

Errors: Raises BadWNum if WIndx invalid Raises WTooBig if window would extend off the screen

Procedure SFullWindow;

Abstract:

Changes the parameters of the current window to be the full screen

SideEffects: Changes the size of the current window. Does NOT refresh or change the title line or erase anything or move the cursor

Procedure RefreshWindow(WIndx: WinRange);

Abstract:

Redraws window outline and title area (but not title text)

Parameters:

Window to refresh (better be already created)

Errors: Raises BadWNum if WIndx undefined

```
Procedure GetWindowParms(var WIndx: WinRange; var OrgX, OrgY, Width, Height:
    integer; var hasTitle: Boolean);

    Abstract:

        Returns parameters for current window

    Parameters:

        All set to current window's values

Procedure ChangeWindow(WIndx: WinRange);

    Abstract:

        Writes out current window's parameters and changes to new one

    Parameters:

        WindX is new window's number

    Errors: Raises BadWNum if WIndx undefined

Procedure SetFont(NewFont: FontPtr);

    Abstract:

        Changes font to be NewFont

    Parameters:

        NewFont is font to use

    SideEffects: Changes font in current window so all further writes
        (including titles) will be in this font

Function GetFont: FontPtr;

    Abstract:

        Returns current font

    Returns: font currently in use

Procedure ScreenReset;

    Abstract:

        Erases screen; Removes all window; sets Window 0 to have full
        screen boundary and a blank title

    SideEffects: Erases or sets all parameters; font set to system font
```

Procedure ScreenInit;

Abstract:

Sets FirstWindP to NIL and sets up default window; NOTE: CALL THIS PROCEDURE ONCE AT SYSTEM INITIALIZE

Calls: ScreenReset;

Procedure Line(Style: LineStyle; X1, Y1, X2, Y2: integer; Origin: RasterPtr);

Abstract:

Draws a line.

Parameters:

Style - function for the line; X1, X2, Y1, Y2 - end points of line.

Origin - pointer to the memory to draw lines in. Use SScreenP for Origin to draw lines on the screen.

Procedure SVarLine(Style: LineStyle; X1, Y1, X2, Y2, Width: integer; Origin: RasterPtr);

Abstract:

Draws a line. Same as Line except it takes the buffer width as a parameter. This is only useful when drawing lines in off-screen buffers.

Parameters:

Style - function for the line; X1, X2, Y1, Y2 - end points of line.

Width - the word width of the "origin" buffer.

Origin - pointer to the memory to draw lines in. Use SScreenP for Origin to draw lines on the screen.

```
module Scrounge;

    Abstract:

        This module contains the procedure Scrounge which allows a small
        amount of debugging.  Since there are no symbol tables or micro--
        code support for breakpoints, you can look at the stack trace and
        examine variables by offsets.  The types of the variables have to
        be specified by the user.

        Written by: Brad A. Myers  1-May-1981

        Copyright (C) 1981,1982  PERQ Systems Corporation.

Version Number V0.27
{//////////////////////} EXPORTS {\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\}

Procedure Scrounge(ES, ER, PStart, PEnd, ExcSeg, RaiseAP: Integer);

Procedure Scrounge(ES, ER, PStart, PEnd, ExcSeg, RaiseAP: Integer);

    Abstract:

        Scrounge is called when uncaught signals are noticed or when the
        user types ^SHIFT-D.  It allows looking around at local and global
        vbles and the stack trace.  If ^SHIFT-D then can continue with
        program, otherwise aborts when exit

    Parameters:
        ES - segment number of exception
        ER - routine number of exception
        PStart - offset of start of parameters to exception
        PEnd - offset of end of parameters to exception
        ExcSeg - the segment number of the exceptions module if (ES =
            ExcSeg) and (ER = ErrDump) then is ^SHIFT-D For now, can't
            tell ^SHIFT-C
        RaiseAP - the offset for AP for Raise itself (caller is person who
            did the raise)
```

```
module Sid;

Sid - Screen Image Dump.
J Strait    15 Mar 83.
Copyright (C) PERQ Systems Corporation, 1983.
```

Abstract:

Several hardcopy options available for the PERQ computer are capable of printing an image of the display screen. A module to print an image of the screen is included with each of these hardcopy options. The interfaces to these modules are identical in order that programs may be written without knowing which hardcopy option is available. This version of Sid is provided for systems with no hardcopy option. Programs which provide hardcopy ability may import Sid and later be linked with the version of Sid which is specific to a hardcopy device.

Every incarnation of Sid should implement all routines included here. This module may be used as a skeleton for other versions.

The comments in every incarnation should specify the name of the device served and a description of the reasons for raising failure:

Device name: Null. Errors raised: SidNone is always raised.

```
Version Number V1.1

exports

type SidWhy = (SidNone,          { the device is not connected }
               SidBroken,        { the device is physically broken }
               SidHelp,          { the device needs human help, e.g. paper
                                   out }
               SidBusy);         { the device is busy: try later }

procedure Sid( Destination: String );
function  SidExplain( Why: SidWhy ): String;
function  SidDevice: String;


exception SidFail( Why: SidWhy );
```

Abstract:

SidFail is raised when the screen image cannot be printed. Resuming from this exception is allowed: resuming from fatal errors (SidNone, SidBroken) exits from Sid and resuming from non-fatal errors (SidHelp, SidBusy) retries.

Parameters:

Why - The reason that the screen image could not be printed. This value may be converted to a character string with the SidExplain function.

```
procedure Sid( Destination: String );
```

Abstract:

The Sid procedure prints an image of the PERQ display screen to a certain hardcopy device.

Parameters:

Destination - A string describing the destination of the hardcopy. For the EtherNet version of Sid this is the string name of the EtherNet Sid server. If non-null, Destination is used as the name of the server machine, and if null, Sid looks in the user profile for an entry of the form #Sid <ServerName> to determine the name of the server machine. If the parameter is null and there is no entry in the user profile, any available server is used. Destination could be defined differently for other versions of Sid. For example, it could be the name of a file to which a screen image is written.

Errors: SidFail is raised if the screen image cannot be printed. This incarnation of Sid always raises SidFail(SidNone).

```
function  SidExplain( Why: SidWhy ): String;
```

Abstract:

SidExplain converts a SidWhy value into a character string.

Parameters:

Why - The value to explain.

Returns: The explanation of Why.

```
function  SidDevice: String;
```

Abstract:

SidDevice returns the string name of the hardcopy device used by this version of Sid.

Returns: The name of the hardcopy device.

```
module Stream;

Stream - Perq Pascal stream package.
John Strait              ca. Jan 80.
Copyright (C) PERQ Systems Corporation, 1980, 1981, 1982, 1983

     Abstract:

          This module implements the low-level Pascal I/O.  It is not
          intended for use directly by user programs, but rather the
          compiler generates calls to these routines when a Reset, Rewrite,
          Get, or Put is encountered.  Higher-level character I/O functions
          (Read and Write) are implemented by the two modules Reader and
          Writer.

          In this module, the term "file buffer variable" refers to F^ for a
          file variable F.

Version Number V1.25

exports

imports FileDefs from FileDefs;

const StreamVersion = '1.23';

      IdentLength = 8;          { significant characters in an identifier }

type pStreamBuffer = ^StreamBuffer;

     StreamBuffer = record case integer of  { element size: }
      0: (W: array[0..255] of integer);       { 1 or more words, or
                                                > 8 bits }
      1: (B1: packed array[0..0] of 0..1);    { 1 bit }
      2: (B2: packed array[0..0] of 0..3);    { 2 bits }
      3: (B3: packed array[0..0] of 0..7);    { 3 bits }
      4: (B4: packed array[0..0] of 0..15);   { 4 bits }
      5: (B5: packed array[0..0] of 0..31);   { 5 bits }
      6: (B6: packed array[0..0] of 0..63);   { 6 bits }
      7: (B7: packed array[0..0] of 0..127);  { 7 bits }
      8: (B8: packed array[0..0] of 0..255);  { 8 bits }
      9: (C: packed array[0..255] of char);   { for character structured }
      end;

     ControlChar = 0..#37;                  { ordinal of an ASCII control
                                              character }

     FileKind = (BlockStructured, CharacterStructured);

     FileType = { file of Thing }
      packed record
       Flag: packed record case integer of
        0: (CharReady  : boolean;            { character is in file window }
            FEoln      : boolean;            { end of line flag }
            FEof       : boolean;            { end of file }
```

```
            FNotReset  : boolean;          { false if a Reset has been
                                             performed on this file }
            FNotOpen   : boolean;          { false if file is open }
            FNotRewrite: boolean;          { set false if a Rewrite has
                                             been performed on this file }
            FExternal  : boolean;          { not used - will be
                                             permanent/temp file flag }
            FBusy      : boolean;          { IO is in progress }
            FKind      : FileKind);
          1: (skip1     : 0..3;
            ReadError : 0..7);
          2: (skip2     : 0..15;
            WriteError: 0..3)
          end;
        EolCh, EofCh, EraseCh, NoiseCh: ControlChar; {self explanatory}
        OmitCh         : set of ControlChar;
        FileNum        : integer;      { POS file number }
        Index          : integer;      { current word in buffer for un-packed
                                         files, current element for packed
                                         files }
        Length         : integer;      { length of buffer in words for un-
                                         packed files, in elements for packed
                                         files }
        BlockNumber    : integer;      { next logical block number }
        Buffer         : pStreamBuffer;{ I/O buffer }
        LengthInBlocks: integer;       { file length in blocks }
        LastBlockLength:integer;       { last block length in bits }
        SizeInWords    : integer;      { element size in words, 0 means
                                         packed file }
        SizeInBits     : 0..16;        { element size in bits for packed
                                         files }
        ElsPerWord     : 0..16;         { elements per word for packed files }
        Element: { Thing } record case integer of  {The File window}
          1: (C: char);
          2: (W: array[0..0] of integer)
          end
        end;

      ChArray = packed array[1..1] of char; {For read/write character array}

      Identifier = string[IdentLength];
      IdentTable = array[0..1] of Identifier;

var StreamSegment: integer;     { Segment buffer for I/O buffers }
    KeyBuffer: packed array[0..255] of char;
    KeyNext, KeyLength: integer;

procedure StreamInit( var F: FileType; WordSize, BitSize: integer;
                      CharFile: boolean );
procedure StreamOpen( var F: FileType; var Name: PathName;
                      WordSize, BitSize: integer; CharFile: boolean;
                      OpenWrite: boolean );
procedure StreamClose( var F: FileType );
procedure GetB( var F: Filetype );
procedure PutB( var F: Filetype );
procedure GetC( var F: Filetype );
```

```
procedure PutC( var F: FileType );
procedure PReadln( var F: Filetype );
procedure PWriteln( var F: Filetype );
procedure InitStream;
function StreamName( var F: FileType ): PathName;
function FullLn( var F: Text ): Boolean;
procedure StreamKeyBoardReset( var F: Text );

exception ResetError( FileName: PathName );
```

Abstract:

Raised when unable to reset a file--usually file not found but also could be ill-formatted name or bad device name.

Parameters:

FileName - name of the file or device.

```
exception RewriteError( FileName: PathName );
```

Abstract:

Raised when unable to rewrite a file--usually file unknown device or partition but also could be ill-formatted name or bad device name.

Parameters:

FileName - name of the file or device.

```
exception NotTextFile( FileName: PathName );
```

Abstract:

Raised when an attempt is made to open a non-text file to a character-structured device.

Parameters:

FileName - name of the device.

```
exception NotOpen;
```

Abstract:

Raised when an attempt is made to use a file which is not open.

```
exception NotReset( FileName: PathName );
```

Abstract:

Raised when an attempt is made to read a file which is open but has not been reset.

Parameters:

FileName - name of the file or device.

```
exception NotRewrite( FileName: PathName );
```

Abstract:

Raised when an attempt is made to write a file which is open but has not been rewritten.

Parameters:

FileName - name of the file or device.

```
exception PastEof( FileName: PathName );
```

Abstract:

Raised when an attempt is made to read past the end of the file.

Parameters:

FileName - name of the file or device.

```
exception UnitIOError( FileName: PathName );
```

Abstract:

Raised when IOCRead or IOCWrite returns an error status.

Parameters:

FileName - name of the device.

```
exception TimeOutError( FileName: PathName );
```

Abstract:

Raised when a device times out.

Parameters:

FileName - name of the device.

```
exception UndfDevice;
```

Abstract:

Raised when an attempt is made to reference a file which is open to a character-structured device, but the device number is bad. In the current system (lacking automatic initialization of file variables), this may be caused by referencing a file which has never been opened.

```
exception NotIdentifier( FileName: PathName );
```

Abstract:

Raised when an identifier is expected on a file, but something else is encountered.

Parameters:

FileName - name of the file or device.

```
exception NotBoolean( FileName: PathName );
```

Abstract:

Raised when a boolean is expected on a file, but something else is encountered.

Parameters:

FileName - name of the file or device.

```
exception BadIdTable( FileName: PathName );
```

Abstract:

Raised by ReadIdentifier when the identifier table is bad.

Parameters:

FileName - name of the file or device.

```
exception IdNotUnique( FileName: PathName; Id: Identifier );
```

Abstract:

Raised when non-unique identifier is read.

Parameters:

FileName - name of the file or device. Id - the identifier which was read.

```
exception IdNotDefined( FileName: PathName; Id: Identifier );
```

Abstract:

Raised when an undefined identifier is read.

Parameters:

FileName - name of the file or device.

Id - the identifier which was read.

```
exception NotNumber( FileName: PathName );
```

Abstract:

Raised when a number is expected on a file, but something else is encountered.

Parameters:

FileName - name of the file or device.

```
exception LargeNumber( FileName: PathName );
```

Abstract:

Raised when a number is read from a file, but it is too large.

Parameters:

FileName - name of the file or device.

```
exception SmallReal( FileName: PathName );
```

Abstract:

Raised when a real number is read from a file, but it is too small.

Parameters:

FileName - name of the file or device.

```
exception BadBase( FileName: PathName; Base: Integer );
```

Abstract:

Raised when an attempt is made to read a number with a numeric base that is not in the range 2..36.

Parameters:

FileName - name of the file or device.

Base - numeric base (which is not in the range 2..36).

```
exception LargeReal( FileName: PathName );
```

Abstract:

Raised when a real number is read from a file, but it is too large.

Parameters:

FileName - name of the file or device.

```
exception RealWriteError( FileName: PathName );
```

Abstract:

Raised when an attempt is made to write a real number which is invalid.

Parameters:

FileName - name of the file or device.

```
exception NotReal( FileName: PathName );
```

Abstract:

Raised when a real number is expected on a file, but something else is encountered.

Parameters:

FileName - name of the file or device.

```
Procedure StartTranscript(fileName: PathName; append: boolean);
Procedure StopTranscript;
Procedure TransChar(c: Char);
```

```
Exception TransError(kind: String);
```

Abstract:

Raised when startTranscript called but a transcript is already open or if StopTranscript is called and no transcript is active. Also, if StartTranscript and file cannot be created.

Parameters:

Kind - a string describing the error.

```
procedure StreamInit( var F: FileType; WordSize, BitSize: integer; CharFile:
    boolean );
```

Abstract:

Initializes, but does not open, the file variable F. Automatically called upon entry to the block in which the file is declared. (To be written when the compiler generates calls to it.)

Parameters:
F - the file variable to be initialized.
WordSize and BitSize are the size of an element of the file.
CharFile - determines whether or not the file is of characters.

```
procedure StreamClose( var F: FileType );
```

Abstract:

Closes the file variable F.

Parameters:

F - the file variable to be closed.

```
procedure StreamOpen( var F: FileType; var Name: PathName; WordSize,
    BitSize: integer; CharFile: boolean; OpenWrite: boolean );
```

Abstract:

Opens the file variable F. This procedure corresponds to both Reset and Rewrite.

Parameters:
F - the file variable to be opened.
Name - the file name.
WordSize - number of words in an element of the file (0 indicates a packed file).
BitSize - number of bits in an element of the file (for packed files).
CharFile - true if the file is a character file.
OpenWrite - true if the file is to be opened for writing (otherwise it is opened for reading).

Errors:
ResetError if unable to reset the file.
RewriteError if unable to rewrite the file.
NotATextFile if an attempt is made to open a non-text file to a character structured device.

```
procedure GetB( var F: Filetype );
```

Abstract:

Advances to the next element of a block-structured file and gets it into the file buffer variable.

Parameters:

F - the file to be advanced.

Errors:
NotOpen if F is not open.
NotReset if F has not been reset.
PastEof if an attempt is made to read F past Eof.

```
procedure GetC( var F: Filetype );
```

Abstract:

Advances to the next element of a character-structured file and gets it into the file buffer variable.

Parameters:

F - the file to be advanced.

Errors:
NotOpen - if F is not open.
NotReset - if F has not been reset.
PastEof - if an attempt is made to read F past Eof.
TimeOutError - if RS: or RSX: times out.
UnitIOError - if IOCRead doesn't return IOEIOC or IOEIOB.
UndfDevice - if F is open, but the device number is bad.

```
procedure PutB( var F: Filetype );
```

Abstract:

Writes the value of the file buffer variable to the block-structured file and advances the file.

Parameters:

F - the file to be advanced.

Errors:
NotOpen - if F is not open.
NotRewrite- if F has not been rewritten.

```
procedure PutC( var F: FileType );
```

Abstract:

Writes the value of the file buffer variable to the character-structured file and advances the file.

Parameters:

F - the file to be advanced.

Errors:
NotOpen - if F is not open.
NotRewrite - if F has not been rewritten.
UnitIOError- if IOCWrite doesn't return IOEIOC or IOEIOB.
TimeOutError- if RS: or RSX: times out.
UndfDevice - if F is open, but the device number is bad.

```
procedure PReadln( var F: Filetype );
```

Abstract:

Advances to the first character following an end-of-line.

Parameters:

F - the file to be advanced.

```
procedure PWriteln( var F: Filetype );
```

Abstract:

Writes an end-of-line.

Parameters:

F - the file to which an end-of-line is written.

```
procedure StreamKeyBoardReset( var F: Text );
```

Abstract:

Clears the keyboard input buffer and the file variable F so that all input typed up to this point will be ignored.

Parameters:

F - file to be cleared.

```
procedure InitStream;
```

Abstract:

Initializes the stream package. Called by System.

```
function FullLn( var F: Text ): Boolean;
```

Abstract:

Determines if there is a full line in the keyboard input buffer. This is the case if a carriage-return has been typed. This function is provided in order that a program may continue to do other things while waiting for keyboard input. If the file is not open to the console, FullLn is always true.

Parameters:

F - file to be checked.

Returns: True if a full line has been typed.

Errors:
NotOpen - if F is not open.
NotReset - if F has not been reset.

```
function StreamName( var F: FileType ): PathName;
```

Abstract:

Returns the file name associated with the file variable F. For block-structured files, the full path name including device and partition is returned. For character-structured files, the device name is returned.

Parameters:

F - file variable whose name is to be returned.

```
Procedure StartTranscript(fileName: PathName; append: boolean);
```

Abstract:

Starts a transcript to the specified file. The transcript will contain all characters written to the screen. A user-typed backspace character will be echoed to the file as a backslash "\". All other characters will be copied directly. Do not leave the transcript on while running screen-based programs like PATCH or the EDITOR since they will input lots of garbage into the transcript. The transcript cannot be read until the StopTranscript routine is called. Note that some programs, notably TYPEFILE, will not output anything to the transcript since they RasterOp to the screen directly. To get a file into the transcript, use COPY file ~ CONSOLE:

***WARNING*** It is VERY dangerous to have transcripting on while

running the Scavenger or any similar program. No checking is done to insure that this is not done. Caveat Emptor. ***WARNING*** Also, it is dangerous to run Typefile while transcripting is on. The data typed out will be wrong (since TypeFile uses ReadDisk) and the the PERQ may crash. ***WARNING***

Parameters:

fileName - the name of the file that the transcript is supposed to go in. If the name is empty or the file cannot be opened then the TransError exception is raised.

append - if false, the file is started from scratch. That is, it will contain only the text from this session. If append is true, then the new text will be put at the end of the file if it already exists. If the file does not exist, append will be identical to not append.

Errors:

TransError - raised if this procedure is called and a transcript is already in effect or if the filename passed is invalid.

Procedure TransChar(c: Char);

Abstract:

Enters a character into the transcript.

Parameters:

c - the character to put into the transcript.

Errors:

TransError - raised if transcript is not open.

Procedure StopTranscript;

Abstract:

Closes the transcript file. You must call this before accessing the transcript. If not, the file will be incomplete and not closed.

Errors:

TransError - raised if no transcript is open.

```
Program System;

Perq Software Group.
Copyright (C) PERQ Systems Corporation, 1980, 1981, 1983.

     Abstract:

          Initialize POS and go into loop alternately running Shell and user
          program
Version Number V2.17

{********************} Exports {********************}

Const MainVersion = 'G';
      DebugSystemInit = False;
      FirstDDS = 199;
      ShellConst = 'Shell.';
      LogConst = 'LogIn.';
      PFileConst = 'Default.Profile';

      SysTiming = True;     { Gather System timing statistics.  If this
                              constant is changed, IO, Loader, Memory,
                              Movemem, System, and Shell should be
                              re-compiled, and the System should be
                              re-linked. }

 Type Sys9s = String[10];

Var UsrCmdLine: String[255];     {Command line entered by user}
    UseCmd: Boolean;             {Set True to tell shell to execute
UsrCmdLine}
    InCmdFile: Boolean;          {True if shell commands from file}
    LastFileName,                {Name of file to use if none given}
    RFileName,                   {Name of next program to run}
    ShellName: String;           {Name of Shell}

    CurUserID,                   {Index of user in System.Users}
    CurGroupID: 0..255;          {Groupid of current user}
    CurUserName,                 {LogIn name of current user}
    CurPFile: String;            {Name of current profile file}
    UserMode: Boolean;           {True while executing user program}

    CtrlCPending: Boolean;       {True if one control-C typed}

    NextSSize: Integer;          {Screen size for next program}
    NextSComplemented: Boolean; {Whether to complement bottom for next pgm}
    NextSOff: Boolean;           {Whether bottom should display data bits}
    DefCursFunct: Integer;       {What to set curs func to after each prog}
    DefScrComp: Boolean;         {Default value for NextSComplemented}
    DefScrOff: Boolean;          {Default value for NextSOff}

    ShellCtrl: pointer;          {Pointer to information record for Shell}
    TimeFID: integer;            {File ID of file holding current time}
    CmdSegment: Integer;         {SegmentNumber of seg holding command files}

    InPmd: Boolean;              {True if in Scrounge (PostMortemDump)}
```

```
    SysDisk: Integer;           {Number of the disk booted from}
    SysBootChar: Integer;       {Ord(char held down to boot)}

    StrVersion: string;         {System version number as a string}
    SystemVersion: Integer;     {Integer giving system version number}
    SystemInitialized: Boolean; {True after system initialized}
    DDS: Integer;               {Keeps current diagnostic display value}
    ShouldReEnableSwapping: Boolean;  {True if swapping must be reenabled}
    SavedSwapId: Integer;       {Save id of where to swap to}

    {$ifc SysTiming then}
    LoadTime, OldLoadTime: long;
    ExecuteTime, OldExecuteTime: long;
    SwapTime, OldSwapTime: long;
    MoveTime, OldMoveTime: long;
    IOTime, OldIOTime: long;
    PrintStatistics: Boolean;
    {$endc}

    UserPtr: pointer;           {A pointer variable for use between user
                                 programs.  (Use IncRefCount to keep
                                 segment)}
    UserInt: integer;           {May be a segment number for UserPtr}

    DemoInt: Integer;           {reserved for Demo system}

    isFloppy: Boolean;          {true if booted from floppy, else false}
    pointAllowed: Boolean;      {true if should use pointing device}

    DefRealRelTablet:boolean;   {true if KrizTablet/BitPad in true relative
                                 mode }
    DefTabletType: integer;     {assigned ord(KrizTablet) by Login etc }

    CurRFileName: String;       {the current run file; used by the symbollic
                                 debugger}

{*** WARNING!!  IF YOU CHANGE THE EXPORTED PROCEDURES AND EXCEPTIONS, MAKE
{***           SURE THE NUMBERS FOR THE FOLLOWING EXCEPTIONS ARE UPDATED
{***           AND RECOMPILE SCROUNGE IF CHANGED !!!!! *****}

{*** WARNING!!  DO NOT CHANGE THE ORDER OF THE ^C EXCEPTIONS !!!!! *****}

Procedure Command;
Procedure SetDDS( Display: Integer );
Procedure SysVers( n: integer; var S: string );

Const ErrCtlC = 4; {******}
```

Exception CtlC;

Abstract:

CtlC is raised by the KeyBoard interrupt routine when a control-c is typed. If you handle this exception you should clear CtrlCPending in your handler. If you are catching control-c's to try to prevent aborts, you should enable CtlCAbort also, since the Stream package will raise it when the control-c is read.

Const ErrCtlCAbort = 5; {******}

Exception CtlCAbort;

Abstract:

CtlCAbort is raised by the KeyBoard interrupt routine when the second of two adjacent control-c's is typed. It is also raised by the Stream package when a control-c is read. If you handle this exception you should clear CtrlCPending in your handler.

When this is raised by the KeyBoard interrupt routine, the KeyBoard type-ahead buffer is cleared. If you want to prevent this, you must catch CtlC also.

If your program uses a Text file and you want to clear the line editing buffer for that file, you should call the Stream routine StreamKeyBoardReset(F) (assuming F is the name of the file). If F is a Text file which is attached to the console, this will get rid of the character F^ points to and clear Stream's line editing buffer.

Const ErrCtlShftC = 6; {******}

Exception CtlShftC;

Abstract:

CtlShftC is raised by the KeyBoard interrupt routine when a control- shift-c is typed. If you handle this exception you should clear CtrlCPending in your handler.

When this is raised by the KeyBoard interrupt routine, the KeyBoard type-ahead buffer is cleared. You cannot prevent this.

If your program uses a Text file and you want to clear the line editing buffer for that file, you should call the Stream routine StreamKeyBoardReset(F) (assuming F is the name of the file). If F is a Text file which is attached to the console, this will get rid of the character F^ points to and clear Stream's line editing buffer.

```
        Const ErrExitProgram = 7; {******}

Exception ExitProgram;
```

Abstract:

ExitProgram is raised to abort (or exit) a program. The default handler for CtlCAbort and Scrounge raise this exception.

WARNING: No one but System and Loader should Handle this exception. Anyone may raise it to exit a program.

```
        Const ErrHelpKey = 8; {******}

Exception HelpKey(var retStr: Sys9s);
```

Abstract:

HelpKey is raised when the HELP key is hit.

Parameters:

retStr - the set of characters to put into the input stream. This should be set by the handler if it continues from the exception. Likely values are "/Help<CR>" and chr(7) (the current value returned). The key board interupt routine sets retStr to '' before raising this exception so if not set, and the handler resumes, nothing will be put into the input stream.

Resume: Allowed. Should set retStr first.

```
        Const ErrHardCopy = 9; {******}

Exception HardCopy;
```

Abstract:

HardCopy is raised when Control-Shift-P is typed. A default handler is provided in System which calls Sid to do a Screen Image Dump. The HardCopy exception may be raised by a program which wishes to print a screen image dump.

Resume: Encouraged.

```
        type DoubleWord = ^integer;     {should use Long instead}
```

Procedure SetDDS( Display: Integer );

Abstract:

SetDDS sets the diagnostic display to a particular value.

Parameters:

Display - Desired value of the diagnostic display.

procedure SysVers( n: integer; var S: string );

Abstract:

This procedure will provide the caller with a string that is the Version number of the current system.

Parameters:

n is the minor version number of the system.

S will be set to the current minor version of the system.

Procedure Command;

Abstract:

This procedure alternately loads Shell and the user programs whose runfile names are generated by Shell. It is invoked by the main program in System and can be exited only if the user types ^C or if a runtime error occurs.

```
module UserPass;

     Abstract:

          This module provides facilities for dealing with the password and
          accounts file for PERQ.  The login and protection facilities for
          Perq provide a very simple user validification.  This system is
          NOT completly secure.

          Written by: Don Scelza

          Copyright (C) PERQ Systems Corporation,  1981.

Version Number V1.4
{********************} Exports {********************}

type IDType  = 0..255;

    PassType =  ^Integer;                { a two word value }

    UserRecord = packed record
            InUse: boolean;                  { is this entry in use. }
            Name: String[31];                { Name of the user }
            UserID: IDType;                  { The user ID of the user. }
            GroupID: IDType;                 { The group ID of the user. }
            EncryptPass: PassType;           { The encrypted password. }
            Profile: String;                 { Path name of the profile
                                               file. }
        end;


function FindUser(UserName: String;  var UserRec: UserRecord): Boolean;
function ValidUser( UserName, Password: String;
                          var UserRec: UserRecord): Boolean;
function AddUser(UserName, Password: String;  Group: IDType;
                 ProPath: String): Boolean;
procedure NewUserFile;
procedure ListUsers;
function RemoveUser(UserName: String): boolean;

const PassFile = '>System.Users';

const MaxUsers = 10;

type Users = array[0..MaxUsers] of UserRecord;
```

function FindUser(UserName: String; var UserRec: UserRecord): Boolean;

Abstract:

This function is used to see if a user exists in the user file.

Parameters:

UserName is the name of the user that we are looking for.

UserRec is a var parameter that is used to return the information about the user UserName if he is in the file.

Results: This procedure will return true if the user UserName was in the user file. It will return False otherwise.

function ValidUser( UserName, Password: String;
var UserRec: UserRecord): Boolean;

Abstract:

Sees if a user name and password match.

Parameters:

Username is the name of the user that we want to check.

Password is the password for the user.

UserRec will be filled with the user information if the user name and password match.

Results: If the password is valid for the user then return true. Otherwise return false.

Side Effects: This function will change the file PassFile.

function AddUser(UserName, Password: String; Group: IDType; ProPath: String): Boolean;

Abstract:

Adds a new user to the user file or changes the parameters of an existing user.

Parameters:

Username is the name of the user to add or change.

Password is the password for the user.

Group is the group number for the new user.

ProPath is the path name of the profile file for this user.

Results: If the user could be added or changed then return true. Otherwise return false.

Side Effects: This function will change the file PassFile.

```
procedure NewUserFile;
```

Abstract:

This procedure is used to create a new user file.

Side Effects: This procedure will create a new file. It will destroy any information in the current file.

```
procedure ListUsers;
```

Abstract:

Supplies a list of the valid users.

```
function RemoveUser(UserName: String): boolean;
```

Abstract:

Removes a user from the list of valid users.

Parameters:

UserName is the name of the user that is to be removed.

Results: If the user could be removed, return true. Otherwise return false.

```
module UtilProgress;

Progress Reporting Routines

Copyright (C) 1981  PERQ Systems Corporation

     Abstract:

          Routines to show progress of utilities.

exports

Procedure LoadCurs;
Procedure ShowProgress(NumLines: Integer);
Procedure QuitProgress;
Procedure StreamProgress( var F: File );
Procedure ComputeProgress( Current, Max: Integer );
Procedure LoadBusy;

Procedure LoadCurs;

     Abstract:

          Sets up the cursor before showing progress.

Procedure LoadBusy;

     Abstract:

          Sets up the cursor so that we can show that we are busy.  In busy
          mode, each ShowProgress moves the cursor by one in a random
          direction.  This should be used when an operation is taking place
          and the utility cannot tell how long until it is done.

Procedure QuitProgress;

     Abstract:

          No more progress to report, turn off the cursor.

     Calls: IOCursorMode.

Procedure ShowProgress(NumLines: Integer);

     Abstract:

          If started by LoadCurs then Indicate progress by moving the cursor
          down a certain number of scan lines.  If started by LoadBusy then
          update busy cursor to show that doing something.

     Parameters:

          NumLines is the number of scan lines to move the cursor.
```

Side Effects:

CursPos is modified.

BusyX is modified if <> -1.

Environment: Assumes LoadCurs or LoadBusy has been called.

Calls: IOSetCursorPos.

Procedure StreamProgress( var F: File );

Abstract:

Indicate progress reading a Stream file.

Parameters:

F is a Stream file which has been Reset.

Side Effects: CursPos is modified.

Calls: IOSetCursorPos.

Errors:

NotOpen if F is not open.

NotReset if F is open but not Reset.

Procedure ComputeProgress( Current, Max: Integer );

Abstract:

Indicate progress given a current and maximum value.

Parameters:

Current is the current value.

Max is the maximum value.

Side Effects: CursPos is modified.

Calls: IOSetCursorPos.

```
module Virtual;

Virtual - Perq virtual memory manager.
J. P. Strait        1 Jan 80.
Copyright (C) PERQ Systems Corporation, 1980, 1982.

    Abstract:

        Virtual is the Perq virtual memory manager.  It supervises the
        segment tables and exports procedures for swapping memory
        segments. Virtual is the portion of the Perq memory manager which
        must remain memory resident at all times. Perq physical memory is
        segmented into separately swappable items (called segments) which
        may contain either code or data.

        Design: See the Q-Code reference manual.

Version Number V3.2

exports

const VirtualVersion = '3.2';

imports Memory from Memory;
imports IO_Unit from IO_Unit;
imports DiskIO from DiskIO;

function  ReturnSegment: SegmentNumber;
procedure ReleaseSegmentNumber( Seg: SegmentNumber );
function  NewSegmentNumber: SegmentNumber;
procedure MakeEdge( var E: MMEdge; S: SegmentNumber );
procedure DeleteSegment( var S: SegmentNumber );
procedure SwapOut( var E: MMEdge );
procedure SwapIn( E: MMEdge; S: SegmentNumber; P: MMPosition );
procedure Compact;
procedure KeepSegments;
procedure FindHole( Fsize: MMIntSize; ForUserSegment: Boolean );
procedure IncIOCount( S: SegmentNumber );
procedure DecIOCount( S: SegmentNumber );
procedure SwapSegmentsIn( S1, S2, S3, S4: SegmentNumber );

var ScreenLast: Integer;
    Keep1, Keep2, Keep3, Keep4: SegmentNumber;
    Kludge: record case Integer of
              1: (A: DiskAddr);
              2: (D: Double)
              end;
    BlockHeader: IOHeadPtr;
    BlockAddress: Double;
    BlockSId: SegId;
    Status: IOStatPtr;
    BootSerialNum: Double;
    BootSegId: SegId;
    SwapSId: SegId;
```

```
function ReturnSegment: SegmentNumber;
```

Abstract:

ReturnSegment finds the segment number of the caller of the procedure which called ReturnSegment by searching the call stack.

Result: ReturnSegment = Segment number of the caller.

Design: This routine depends on the Perq running a single process operating system where the caller is in the same process as the memory manager.

```
procedure ReleaseSegmentNumber( Seg: SegmentNumber );
```

Abstract:

ReleaseSegmentNumber releases a segment number to the list of segment numbers which are not in use.

Parameters:

Seg - Segment number to return to the segment number free list.

```
function NewSegmentNumber: SegmentNumber;
```

Abstract:

NewSegmentNumber allocates the next unused segment number.

Errors: NoFreeSegments if there are no unused segment numbers.

```
procedure MakeEdge( var E: MMEdge; S: SegmentNumber );
```

Abstract:

MakeEdge makes an MMEdge record which the head field set to a certain segment number and the tail field set to the previous segment number (in physical address order).

Parameters:
E - MMEdge record to build.
S - Segment to put in the head field.

Errors: EdgeFailure if MakeEdge can't find the previous segment number.

```
procedure DeleteSegment( var S: SegmentNumber );
```

Abstract:

DeleteSegment returns a segment to the free memory list. This is done (for example) when the segment's reference and IO counts both reach zero.

Parameters:

S - Number of the segment to be destroyed. To facilitate segment table scanning loops that contain calls to DeleteSegment:

* If S was resident, it is changed to be the number of the segment which represents the free memory. This may not be the same as the original value if the original segment is coalesced with an adjacent free segment.

* If S was not resident, it is changed to be the number of the segment which preceded it in the segment table.

* MMFirst is set to have the same value as S on exit.

```
procedure SwapOut( var E: MMEdge );
```

Abstract:

SwapOut swaps a data segment out to disk.

Parameters:

E - An edge where the head is the segment to be swapped and the tail is the previous segment.

Result: E.T and E.H both are set to the number of the new free segment.

Errors:

PartNotMounted if the swapping partition is not mounted.

SwapError if attempt to swap segment out while swapping is disabled.

```
procedure SwapIn( E: MMEdge; S: SegmentNumber; P: MMPosition );
```

Abstract:

SwapIn swaps a segment in from disk.

Parameters:

E - An edge describing where to put the segment in memory. The head is a free segment which will be filled by the segment to be swapped in. The tail is the previous segment.
S - The segment to swap in.
P - The position (low end or high end) to use within the head

segment of the edge.

Errors: SwapInFailure if attempt to swap in a segment which was never swapped out.

procedure Compact;

Abstract:

Compact compacts physical memory by moving as many segments as possible toward low addresses. System segments (those with a reference count greater than one) will not be moved into the screen area, as segments cannot jump over one another.

Errors: CantMoveSegment if attempt to move a segment with non-zero IO count.

procedure KeepSegments;

Abstract:

KeepSegments marks the segments Keep1 through Keep4 as not RecentlyUsed so that they won't be swapped out.

procedure FindHole( Fsize: MMIntSize; ForUserSegment: Boolean );

Abstract:

FindHole attempts to find a hole (free memory) of a certain size. It performs a first-fit search. If a hole cannot be found, memory is compacted, and another first-fit search is performed. Eventually, a swap-out pass will be performed.

Parameters:

Fsize - Minimum size of the hole. This is an internal size--Fsize=n means n+1 blocks.

ForUserSegment - True iff this hole is to be used for a user segment. System segments may not be allocated in the screen area.

procedure IncIOCount( S: SegmentNumber );

Abstract:

IncIOCount increments the count of input/output references to a data segment. A non-zero IO count prevents a segment from being moved, swapped, or destroyed.

IncIOCount will increment the count of only one segment and thus should not be applied to the base segment number of a heap. The segment number should be extracted from the pointer being used.

Parameters:

S - Segment number.

Errors:

UnusedSegment if S is not in use.

FullMemory if S is not resident and there isn't enough memory to swap it in.

```
procedure DecIOCount( S: SegmentNumber );
```

Abstract:

DecIOCount decrements the IO count of a data segment by one. If the reference and IO counts both become zero:

* if the segment is a data segment, it is destroyed.

* if the segment is a code segment, it is destroyed only if it is in the screen or is non-resident.

DecIOCount will decrement the count of only one segment and thus should not be applied to the base segment number of a heap. The segment number should be extracted from the pointer being used.

Parameters:

S - Number of the segment.

Errors: UnusedSegment if S is not in use.

```
procedure SwapSegmentsIn( S1, S2, S3, S4: SegmentNumber );
```

Abstract:

SwapSegmentsIn ensures that when it returns, S1, S2, S3, and S4 are resident.

Parameters:

S1, S2, S3, S4 - segments to swap in.

Errors:

NilPointer if one of the segments is zero.

UnusedSegment if one of the segments is not really in use.

FullMemory if there isn't enough memory to swap one of the segments in.

```
module VolumeSystem;

VolumeSystem - TV. ( Tony Vezza )

CopyRight (C) 1983, PERQ Systems Corporation.

    Abstract:

        This module provides uniform abstractions of the disks available
        on a Perq.  Disks are named by unique Constants of an eneumerated
        Type exported by the module. A set of operations is provided and
        each is named by a Constant of an another enumerated Type exported
        by the module. Each disk is made to appear as an array of pairs of
        data blocks and logical headers.  A general address Type with two
        components, one to specify a disk and another to specify an index
        into the array on that disk, is defined and exported. Mounting and
        dismounting of disks is supported by a pair of Procedures.
        (Mounting a disk means reading a symbolic name from a known
        address on that disk and Recording a mapping of that symolic name
        to an identifier for the disk.) Operations are provided to
        determine the number of pages (pairs of data blocks and logical
        headers) and the last valid address on a given disk.

        Naming conventions (necessitated by short identifier limits):

        "Int" - means "Internal" (hard disk without removable packs).
        "Ext" - means "External" (hard disk with removable packs).
        "Flp" - means "Floppy".
        "Mic" - means "Micropolis".
        "Phy" - means "Physical".
        "Vol" - means "Volume".
        "ID"  - means "identifier" and refers to an abstract name Type.

Version Number V4.8
{******************} exports {**************************}

Imports IOErrors From IOErrors;

Type

 { Values of Type DiskKinds denote distinct classes of disk devices
   which can be connected to a Perq. }

 DiskKinds = (FlpDisk, IntDisk, ExtDisk);

 { FlpUnitNumber, IntUnitNumber, and ExtUnitNumber are Types
   for numbers denoting physical units of each distinct class of disk.
   Separate Types are defined to express the necessity of performing
   distinct run time checks on values of unit numbers for each class of
   disk.  Precise ranges cannot be specified at compile time (i.e. in
   this program text) because of the requirement that this program must
   run on machines of many possible configurations without recompilation. }

 Cardinal = 0 .. #77777;
```

```
FlpUnitNumber = Cardinal;
IntUnitNumber = Cardinal;
ExtUnitNumber = Cardinal;

{ Values of Type PhyDiskID are used to uniquely identify disk units
  for the volume mounting and dismounting operations. }

PhyDiskID = Record
              Case Kind : DiskKinds Of
                 FlpDisk : (FlpUnit : FlpUnitNumber);
                 IntDisk : (IntUnit : IntUnitNumber);
                 ExtDisk : (ExtUnit : ExtUnitNumber)
            End;

{ InternalDiskKinds is the Type whose values denote the various kinds of
  internal disks. }

 IntDiskKinds = (Shugart14, Mic8, Mic5, unsupported);

{ OnVolAddress is used to represent the logical address of a block
  of a volume in data structures on THAT volume or on a nonremovable
  volume; these are typically hints which link multiple file structures
  together.  SOLAR requires that such hints be two word objects with
  the two high order bits set and the eight low order bits cleared.
  The remaining 22 bits are divided into two fields: a 3 bit logical
  volume specifier and a 19 bit volume relative logical block number.
  Certain volume specifiers refer to the nonremovable volumes which are
  mounted as the corresponding logical disk (see comments below for Type
  VolID.  The remaining possible values of a volume specifier field all
  denote the volume that the hint itself is written on. }

OnVolAddress = Long;

{ VolName is a string used as part of full file names to name mounted
  file system disks. }

VolName = String[8];

{ MaxTotalVols is the upper limit on the number of file system volumes
  that can be mounted at one time.  MinVolID and MaxVolID delimit
  the range of non-nil (i.e. actually mounted) file system volumes.
  NilVolID denotes a volume different from any possible mounted volume. }

Const
   MaxTotalVols = 8;
   MaxVolID  = MaxTotalVols - 1;
   MinVolID  = 0;
   NilVolID  = MinVolID  - 1;

{ VolID uniquely identifies a mounted file system volume or a nil
  volume; it is also used as a component of a VolAddress.  VolID values
  in the subrange 0 .. MaxInternalUnits - 1 always denote a nonremovable
  file system volume, i.e. an internal class disk device.  (This is related
  to the requirement that internal physical disks be mounted at
  corresponding fixed VolIDs by the VolMount Function.)  Values in
```

```
  MaxInternalUnits .. MaxVolID denote mounted removeable
  volumes.  MaxInternalUnits is an implicit Constant whose value is
  determined by the configuration of the machine.

  VolRangeType is intended for use as an index Type for arrays which
  correspond to mounted actual disks only. }

Type
  VolID        = NilVolID .. MaxVolID;
  VolRangeType = MinVolID .. MaxVolID;

{ VolBlockNumber is a subrange of Long used to uniquely specify a block of
  a file system volume.  A better Type definition (were it expressible
  in current Perq Pascal) would be: #0 .. #1777777, i.e. 19 bit non-
  negative Integers. }

VolBlockNumber = Long;

{ VolAddress uniquely specifies a block on any of the mounted file
  system disks. }

VolAddress = Record
               Volume      : VolID;
               BlockNumber : VolBlockNumber;
             End;

{ VolBuffer defines an uninterpreted structure to be used for input
  and output buffers for data blocks during volume io operations.
  These buffers must be aligned on 256 word boundaries. }

VolBuffer    = Packed Array[0 .. 4095] Of Boolean;

ptrVolBuffer = ^VolBuffer;

{ VolHeaderBuffer defines an structure to be used for translated input
  and output buffers for header blocks during volume io operations. }

VolHeaderBuffer = Record
                    SerialNumber       : VolBlockNumber;
                    SegmentBlockNumber : Integer;
                    FreeListHint       : Integer; {formerly called filler}
                    PreviousBlock,
                    NextBlock          : VolBlockNumber;
                  End; { VolHeaderBuffer }

PtrVolHeaderBuffer = ^VolHeaderBuffer;

{ VolIOCommand enumerates the commands available in the volume io
  operations, VolIO and TryVolIO. }

VolIOCommand = (VolRd, VolRdCheck, VolWr, VolWrCheck, VolReset,
                { Last two for error reporting only (floppy only) }
                VolHdrRead, VolHdrWrite
               );

procedure InitVolumeSystem ;
```

```
Procedure VolDiskReset( VID : VolID);
Function GetIntDiskKind : IntDiskKinds;
Function VolMount( PID : PhyDiskID ) : VolID;
Procedure VolDisMount( PID : PhyDiskID );
Function VolIDLookUp( Name : VolName) : VolID;
Function VolNameLookUp( VID : VolID) : VolName;
Function VolToOnVolAddr( VA : VolAddress) : OnVolAddress;
Function OnVolToVolAddr( VID  : VolID;
                         OVA  : OnVolAddress ) : VolAddress;
Function VolIDToPhyID( VID : VolID) : PhyDiskID;
Function PhyIDToVolID( PID : PhyDiskID) : VolID;
Function LastVolAddress( VID : VolID ) : VolAddress;
Function VolNumberPages( VID : VolId) : VolBlockNumber;
Procedure VolIO( VA          : VolAddress;
                 Ptr         : PtrVolBuffer;
                 HPtr        : PtrVolHeaderBuffer;
                 VolCommand  : VolIOCommand ) ;

Function TryVolIO( VA          : VolAddress;
                   Ptr         : PtrVolBuffer;
                   HPtr        : PtrVolHeaderBuffer;
                   VolCommand  : VolIOCommand;
                   NumTries    : Integer ) : Boolean;

Exception NoSuchNameForVol(N : VolName);

     Raised by VolIdLookUp(n) if no mounted Volume has N as its name

Exception VBNOutOfRange(VID : VolID; VBN : VolBlockNumber);

     Raised when a VolAddress, VA, passed to an operation is such that
       VA.BlockNumber is greater than VolNumberPages(VA.Volume)

Exception NoSuchVol(vid : VolID);

     Raised when a VolAddress, VA, or a VolID, VID, passed to an operation
       is such that VA.Volume or VID denote a Volume which is not Mounted

Exception NoSuchDevice(D : PhyDiskID);

     Raised when a PhyDiskID, D, passed to an operation denotes a Disk not
     in the
       configuration

Exception VolErrInc(Error_code : Integer {IOEFirstError .. IOELastError} );

     Raised whenever an entry in VolErrorCnt is incremented;
     this is a temporary measure to allow the compatibility version of
     DiskIO to keep its variable ErrorCnt updated.
```

```
Exception VolIOFailure(Msg       : String;
                       Operation : VolIOCommand;
                       Addr      : VolAddress;
                       SoftStat  : Integer);

Exception VolDiskError(Msg : String);

Exception VMountErr( D : PhyDiskID);

    Var VolErrorCnt : Array[IOEFirstError..IOELastError,
                            VolRangeType] Of Integer;

Procedure InitVolumeSystem ;
```

Abstract:

Initialize Volume System

Results: Currently Mounts Volumes - Floppy and HardDisk.

Calls:
- VInitialize

```
Function GetIntDiskKind : IntDiskKinds;
```

Abstract:

Tells What Type of disks are connected to the Internal Disk Controller one of the three values:

- Shugart14
- Mic8
- Mic5

Parameters:

None. Just returns the Value of the Variable, IntDiskType, which is set up at InitVolumeSystem.

Results: Returns the value Internal Disk Kind.

```
Function VolIDLookUp( Name : VolName) : VolID;
```

Abstract:

Scan the Disk Control Array for a Volume with Name and return that Volume's ID.

Parameters:

Name is a Volume Name (a String[8]).

Results: The Volume ID corresponding to the Volume with Name if such a Volume exists and is mounted (in the DCA). If no Volume is mounted then return the Nil Volume ID.

Errors:

- NoSuchNameForVol.

Function VolNameLookUp( VID : VolID) : VolName;

Abstract:

This Function returns the Volume Name of a Volume ID.

Parameters:

VID - VolID.

Results: VolName corresponding to the VID. The VolumeName for a mounted Volume is set up by the Volume Mount Function.

Errors:

- NoSuchVol, Raised if this VID is not Mounted.

Function VolToOnVolAddr( VA : VolAddress) : OnVolAddress;

Abstract:

Result given, in general, by

```
-       OnVolAddress :=  2^31 + 2^30
-                         + VA.Volume * 2^27
-                         + VA.BlockNumber * 2^8
```

Note that if VA.Volume > 3, use 7.

Parameters:

VA - VolAddress

Results:

- OnVolAddress, given by formula above.

Calls:
- VolToLogAddr
- VolNumberPages

Errors:
- NoSuchVol (in VolNumberPages)
- VBNOutofRange

Function OnVolToVolAddr( VID : VolID; OVA : OnVolAddress ) : VolAddress;

Abstract:

Result given by

- VolAddress.Volume := VID
- VolAddress.BlockNumber := OVA Bits <26:8>

Note that the other bits of OVA are Ignored.

Parameters:

OVA - OnVolAddress.

Results:

- VolAddress, given by formula above.

Calls:
- LogToVolAddr
- VolNumberPages

Errors:
- NoSuchVol (in VolNumberPages)
- VBNOutofRange

Function VolIDToPhyID( VID : VolID) : PhyDiskID;

Abstract:

Takes a Volume ID and returns that Volumes Physical Disk ID.

Parameters:

VID - VolID.

Results:

- PhyDiskID selected by given Volume ID.

Errors:

- NoSuchVol

Function PhyIDToVolID( PID : PhyDiskID) : VolID;

Abstract:

Takes a Physical Disk ID and returns that Disk's Volume ID.

Parameters:

PID - PhyDiskID.

Results:

- VolID of given Physical Disk ID.

Errors:

- NoSuchDevice

```
Function LastVolAddress( VID : VolID ) : VolAddress;
```

Abstract:

Returns the Volume Address of the last Sector on the selected disk. Given by:

```
-       LastVolAddress.Volume := VID
-       LastVolAddress.BlockNumber := VolNumberPages - 1
```

Parameters:

VID - VolID

Results:

LastVolAddress - VolAddress

Calls:

- VolNumberPages

Errors:

- NoSuchVol

```
Function VolNumberPages( VID : VolId) : VolBlockNumber;
```

Abstract:

Returns the total number of Blocks on the Volume. Given by:

```
- VolNumberPages := ( PtrDCA^[ VID].PhysParameters.Cylinder
-                  * PtrDCA^[ VID].PhysParameters.Head
-                  * PtrDCA^[ VID].PhysParameters.Sector )
-                  - PtrDCA^[ VID].PhysParameters.BootSize
```

Parameters:

VID - VolID

PhyParameters in DCB
- Cylinder ( Number of Cylinder on Disk.)
- Head ( Number of Tracks per Cylinder.)
- Sector ( Number of Sectors per Track.)
- BootSize ( Number of Sectors for Boot.)

Results:

VolNumberPages - VolBlockNumber ( Long)

Errors:

- NoSuchVol

```
Procedure VolIO( VA          : VolAddress;
     Ptr         : PtrVolBuffer;
     HPtr        : PtrVolHeaderBuffer;
     VolCommand  : VolIOCommand ) ;
```

Abstract:

This routine is used by the File System to perform Disk IO. It calls DoVolIO with a retry count of 15.

Parameters:
VA - VolAddress
Ptr - PtrVolBuffer
HPtr - PtrVolHeaderBuffer
VolCommand - VolIOCommand

Results: The Buffers are either Written onto the Disk, or Disk Data is read into the Buffers.

Calls:

- DoVolIO

Errors:

- ( See DoVolIO.)

```
Function TryVolIO( VA          : VolAddress;
     Ptr         : PtrVolBuffer;
     HPtr        : PtrVolHeaderBuffer;
     VolCommand  : VolIOCommand;
     NumTries    : Integer ) : Boolean;
```

Abstract:

This Function is used by the File System to perform Disk IO. It calls DoVolIO with a retry count of NumTries.

Parameters:
VA - VolAddress
Ptr - PtrVolBuffer
HPtr - PtrVolHeaderBuffer
VolCommand - VolIOCommand
NumTries - Integer

Results:
TryVolIO - Boolean. Indicates whether or not transfer was completed successfully.
- The Buffers are either Written onto the Disk, or Disk Data is read into the Buffers.

Calls: DoVolIO

Errors: ( See DoVolIO.)

```
Function VolMount( PID : PhyDiskID ) : VolID;
```

Abstract:

The Mount Procedure is used to create a DCB for a particular drive and enter that DCB in the Disk Control Array. Returns the VID of the DCA entry which was used to Mount the Disk.

Parameters:

PID - PhyDiskID

Results: VolMount - VolID

Calls: Mount

Errors: ( See Mount.)

```
Procedure VolDisMount( PID : PhyDiskID );
```

Abstract:

Volume DisMount

The DisMount Procedure Disolves the DCB for a particular Drive that was previously mounted, and frees up the Disk Control Array entry which was allocated for that DCB.

Parameters:

PID - PhyDiskID

Results: DisMounts the Disk.

Calls: Dismount

Errors: ( See DisMount.)

Procedure VolDiskReset( VID : VolID);

Abstract:

Used to Reset and Initialize the Disk Controller, Disk Drive and Disk uCode. Drive is Recalibrated.

Parameters:

VID - VolID. Of Disk to be affected.

Results: Drive and Controller is reset and reclibrated.

Calls: UnitIO

Errors: NoSuchVol and VolIOFailure

```
module Writer;

Writer - Stream package output conversion routines.
J. P. Strait   ca. 1 Jan 81.
Copyright (C) PERQ Systems Corporation, 1981.

    Abstract:

        Writer is the character output module of the Stream package.  It
        is called by code generated by the Pascal compiler in response to
        a Write or Writeln.  It is one level above Module Stream and uses
        Stream's output routines.

Version Number V2.2

exports

imports Stream from Stream;

procedure WriteBoolean( var F: FileType; X: Boolean; Field: integer );
procedure WriteCh( Var F: FileType; X: char; Field: integer );
procedure WriteChArray( var F: FileType; var X: ChArray;
                        Max, Field: integer );
procedure WriteIdentifier( var F: FileType; X: integer;
                           var IT: IdentTable; L, Field: integer );
procedure WriteInteger( var F: FileType; X: integer; Field: integer );
procedure WriteString( var F: FileType; var X: String; Field: integer );
procedure WriteX( var F: FileType; X, Field, B: integer );

procedure WriteBoolean( var F: FileType; X: Boolean; Field: integer );

    Abstract:

        Writes a boolean in fixed format.

    Parameters:
        X - the boolean to be written.
        F - the file into which X is to be written.
        Field - the size of the field into which X is to be written.

procedure WriteCh( var F: FileType; X: char; Field: integer );

    Abstract:

        Writes a character in a fixed format.

    Parameters:
        X - the character to be written.
        F - the file into which X is to be written.
        Field - the size of the field into which X is to be written.
```

```
procedure WriteChArray( var F: FileType; var X: ChArray;
                            Max, Field: integer );
```

Abstract:

Writes a packed character array in fixed format.

Parameters:
X - the character array to be written.
F - the file into which X is to be written.
Field - the size of the field into which X is to be written.
Max - the declared length of X.

```
procedure WriteIdentifier( var F: FileType; X: integer;
    var IT: IdentTable; L, Field: integer );
```

Abstract:

Writes an identifier from a table in fixed format.

Parameters:
X - the ordinal of the identifier in the range 0 to L.
F - the file to which X is written.
IT - the table of identifier names indexed from 0 to L.
L - the largest identifier ordinal defined by the table.
Field - the size of the field into which X is written.

Errors: BadIdTable if the length of identifier table is less than 1.

```
procedure WriteInteger( var F: FileType; X: integer; Field: integer );
```

Abstract:

Writes a decimal integer in fixed format.

Parameters:
X - the integer to be written.
F - the file into which X is to be written.
Field - the size of the field into which X is to be written.

```
procedure WriteString( var F: FileType; var X: String; Field: integer );
```

Abstract:

Writes a string in fixed format.

Parameters:
X - the string to be written.
F - the file into which X is written.
Field - the size of the field into which X is written.

```
procedure WriteX( var F: FileType; X, Field, B: integer );
```

Abstract:

Writes an integer in fixed format with base B.

Parameters:

X - the integer to be written.
F - the file into which X is to be written.
Field - the size of the field into which X is to be written.
B - the base of X. It is an integer whose absolute value must be between 2 and 36, inclusive.

Errors: BadBase if the base is not in 2..36.

## Index of Exported Types, Variables, and Constants

| | | |
|---|---|---|
| Cf_MonitorType | TYPE | [ Module Configuration ] |
| Cf_RS232MaxSpeed | VAR | [ Module Configuration ] |
| Cf_RS232Ports | VAR | [ Module Configuration ] |
| Cf_WCSSize | VAR | [ Module Configuration ] |
| CharFile | VAR | [ Module Stream ] |
| ChArray | TYPE | [ Module Stream ] |
| CImpInfo | TYPE | [ Module Code ] |
| CirBufItem | TYPE | [ Module IO_Private ] |
| CirBufPtr | TYPE | [ Module IO_Private ] |
| CirBufSize | CONST | [ Module IO_Private ] |
| CircularBuffer | TYPE | [ Module IO_Private ] |
| Clock | CONST | [ Module IO_Unit ] |
| ClockStat | TYPE | [ Module IO_Unit ] |
| ClockVersion | CONST | [ Module Clock ] |
| CmdArray | TYPE | [ Module CmdParse ] |
| CmdChar | CONST | [ Module CmdParse ] |
| CmdFileChar | CONST | [ Module CmdParse ] |
| CmdListRec | TYPE | [ Module CmdParse ] |
| CmdPVersion | CONST | [ Module CmdParse ] |
| CmdSegment | VAR | [ Program System ] |
| CodeVersion | CONST | [ Module Code ] |
| ComFile | CONST | [ Module FileTypes ] |
| CommentLen | CONST | [ Module Code ] |
| Complex | TYPE | [ Module ComplexFunctions ] |
| ControlChar | TYPE | [ Module Stream ] |
| CString | TYPE | [ Module CmdParse ] |
| CtrlC | CONST | [ Module IOKeyboard ] |
| CtrlCPending | VAR | [ Program System ] |
| CtrlQ | CONST | [ Module IOKeyboard ] |
| CtrlS | CONST | [ Module IOKeyboard ] |
| CurGroupID | VAR | [ Program System ] |
| CurPatPtr | TYPE | [ Module IO_Others ] |
| CurPFile | VAR | [ Program System ] |
| CurRFileName | VAR | [ Program System ] |
| CursF | VAR | [ Module IOVideo ] |
| CursFunction | TYPE | [ Module IO_Others ] |
| CursMode | TYPE | [ Module IO_Others ] |
| Cursor | VAR | [ Module IOVideo ] |
| CursorFile | CONST | [ Module FileTypes ] |
| CursorPattern | TYPE | [ Module IO_Others ] |
| CursorSeg | CONST | [ Module Memory ] |
| CursorX | VAR | [ Module IOVideo ] |
| CursorY | VAR | [ Module IOVideo ] |
| CursType | TYPE | [ Module PopUpCurs ] |
| CurUserID | VAR | [ Program System ] |
| CurUserName | VAR | [ Program System ] |
| CurWind | VAR | [ Module Screen ] |
| CurWindp | VAR | [ Module Screen ] |
| Cyl | TYPE | [ Module DiskDef ] |
| Cylinder-Head-Sector | VAR | [ Module DiskIO ] |
| DataAvailable | VAR | [ Module IO_Private ] |
| DatFile | CONST | [ Module FileTypes ] |
| DirFile | CONST | [ Module FileTypes ] |
| DBLINDSIZE | CONST | [ Module DiskIO ] |
| DBLZERO | CONST | [ Module FileDefs ] |
| DBpFT | CONST | [ Module DiskDef ] |

| | | |
|---|---|---|
| DCBStatus | TYPED | [ Module DiskDef ] |
| DCBStack | VAR | [ Module EtherInterrupt ] |
| DDS | VAR | [ Program System ] |
| DebugSystemInit | CONST | [ Program System ] |
| DefaultCursor | VAR | [ Module IO_Others ] |
| DefaultDeviceName | VAR | [ Module FileDir ] |
| DefaultPartitionName | VAR | [ Module FileDir ] |
| DefCursFunct | VAR | [ Program System ] |
| DefHeapSize | CONST | [ Module Code ] |
| DefIncHeap | CONST | [ Module Code ] |
| DefIncStack | CONST | [ Module Code ] |
| DefRealRelTablet | VAR | [ Program System ] |
| DefScrComp | VAR | [ Program System ] |
| DefScrOff | VAR | [ Program System ] |
| DefStackSize | CONST | [ Module Code ] |
| DefTabletType | VAR | [ Program System ] |
| DemoInt | VAR | [ Program System ] |
| DeviceRecord | TYPE | [ Module AllocDisk ] |
| DeviceType | TYPE | [ Module DiskIO ] |
| DevStatusBlock | TYPE | [ Module IO_Unit ] |
| DevTblEntry | TYPE | [ Module IO_Private ] |
| DevTypes | TYPE | [ Module FTPUtils ] |
| Dev_AckReceived | CONST | [ Module IO_Private ] |
| Dev_Attention | CONST | [ Module IO_Private ] |
| Dev_DataAvailable | CONST | [ Module IO_Private ] |
| Dev_Micropolis | CONST | [ Module IO_Private ] |
| Dev_NakReceived | CONST | [ Module IO_Private ] |
| Dev_ScreenUpdate | CONST | [ Module IO_Private ] |
| Dev_Shugart | CONST | [ Module IO_Private ] |
| Dev_SMD | CONST | [ Module IO_Private ] |
| Dev_StatReceived | CONST | [ Module IO_Private ] |
| Dev_Unused | CONST | [ Module IO_Private ] |
| DIBAddress | CONST | [ Module DiskDef ] |
| DIBlock | TYPE | [ Module DiskDef ] |
| DirBlk | TYPE | [ Module FileSystem ] |
| DIRECTSIZE | CONST | [ Module DiskIO ] |
| DirEntry | TYPE | [ Module DiskIO ] |
| DiskAddr | TYPE | [ Module FileDefs ] |
| DISKBITS | CONST | [ Module DiskIO ] |
| DiskBuffer | TYPE | [ Module DiskIO ] |
| DISKBUFSIZE | CONST | [ Module FileDefs ] |
| DiskCheatType | TYPE | [ Module DiskIO ] |
| DiskCommand | TYPE | [ Module DiskIO ] |
| DiskCtrlBlock | TYPE | [ Module DiskDef ] |
| DiskKinds | TYPE | [ Module VolumeSystem ] |
| DiskSegment | VAR | [ Module DiskIO ] |
| DiskTable | VAR | [ Module AllocDisk ] |
| DiskType | TYPE | [ Module DiskDef ] |
| Double | TYPE | [ Module SystemDefs ] |
| DoubleWord | TYPE | [ Program System ] |
| DskBlockSize | CONST | [ Module DiskDef ] |
| DskCmds | TYPE | [ Module DiskDef ] |
| DskCtrlArray | TYPE | [ Module DiskDef ] |
| DskCyls | CONST | [ Module DiskDef ] |
| DskExHds | CONST | [ Module DiskDef ] |
| DskHds | CONST | [ Module DiskDef ] |

| | | |
|---|---|---|
| DskResult | TYPE | [ Module IO_Unit ] |
| DskSPC | CONST | [ Module DiskDef ] |
| DStatus | TYPE | [ Module DiskDef ] |
| DumpCh | CONST | [ Module IOKeyboard ] |
| e | VAR | [ Module PasReal ] |
| EBoardIO | CONST | [ Module Ether10IO ] |
| EBoardOption | CONST | [ Module Ether10IO ] |
| EIODisk | CONST | [ Module IO_Unit ] |
| EIODskCtrlBlock | TYPE | [ Module DiskDef ] |
| EIOFlag | VAR | [ Module DiskDef ] |
| EP_Ethernet | CONST | [ Module IO_Private ] |
| EP_GetChar | CONST | [ Module IO_Private ] |
| EP_GetCircBuffer | CONST | [ Module IO_Private ] |
| EP_HardDisk | CONST | [ Module IO_Private ] |
| EP_IOStart | CONST | [ Module IO_Private ] |
| EP_PutCircBuffer | CONST | [ Module IO_Private ] |
| EP_ReadCause | CONST | [ Module IO_Private ] |
| EP_ReadTimer | CONST | [ Module IO_Private ] |
| EP_SetEnableMask | CONST | [ Module IO_Private ] |
| EP_UcodeMsg | CONST | [ Module IO_Private ] |
| EP_Z80Msg | CONST | [ Module IO_Private ] |
| ErrCtlC | CONST | [ Program System ] |
| ErrCtlCAbort | CONST | [ Program System ] |
| ErrCtlShftC | CONST | [ Program System ] |
| ErrExitProgram | CONST | [ Program System ] |
| ErrHelpKey | CONST | [ Program System ] |
| ErrorCnt | VAR | [ Module DiskIO ] |
| ErrorType | TYPE | [ Module CmdParse ] |
| ErrStatus | TYPE | [ Module FTPUtils ] |
| Ether10 | CONST | [ Module IO_Unit ] |
| Ether10MBaud | CONST | [ Module SystemDefs ] |
| Ether3 | CONST | [ Module IO_Unit ] |
| Ether3MBaud | CONST | [ Module SystemDefs ] |
| EtherAddress | TYPE | [ Module Ether10IO ] |
| EtherAdRec | TYPE | [ Module Ether10IO ] |
| EtherBuffer | TYPE | [ Module Ether10IO ] |
| EtherCommand | TYPE | [ Module Ether10IO ] |
| EtherDCB | TYPE | [ Module Ether10IO ] |
| EtherHeader | TYPE | [ Module Ether10IO ] |
| EtherRegSave | TYPE | [ Module Ether10IO ] |
| EtherStatus | TYPE | [ Module Ether10IO ] |
| ExceptVersion | CONST | [ Module Except ] |
| ExcSeg | VAR | [ Module Except ] |
| ExDirFile | CONST | [ Module FileTypes ] |
| ExecuteTime | VAR | [ Program System ] |
| ExtFile | CONST | [ Module FileTypes ] |
| ExtUnitNumber | TYPE | [ Module VolumeSystem ] |
| FastEType | CONST | [ Module FTPUtils ] |
| FFS | CONST | [ Module DiskDef ] |
| FHeadPtr | VAR | [ Module DiskDef ] |
| FHpS | CONST | [ Module DiskDef ] |
| FIBlk | CONST | [ Module FileSystem ] |
| FIdS | CONST | [ Module DiskDef ] |
| Field | VAR | [ Module Writer ] |
| file system volume | CONST | [ Module VolumeSystem ] |
| FileID | TYPE | [ Module FileSystem ] |

| | | |
|---|---|---|
| FileID; | VAR | [ Module FileSystem ] |
| FileID; | VAR | [ Module FileSystem ] |
| FileKind | TYPE | [ Module Stream ] |
| FileLength | CONST | [ Module Code ] |
| FileName | VAR | [ Module FileUtils ] |
| FILESPERDIRBLK | CONST | [ Module DiskIO ] |
| FileType | TYPE | [ Module Stream ] |
| FillerSemantics | TYPE | [ Module DiskIO ] |
| FirstBlk | CONST | [ Module FileSystem ] |
| FirstDB | CONST | [ Module DiskDef ] |
| FirstDDS | CONST | [ Program System ] |
| FirstFC | CONST | [ Module DiskDef ] |
| FirstSeg | CONST | [ Module RunWrite ] |
| FirstSeg | VAR | [ Module RunRead ] |
| FirstSystemSeg | VAR | [ Module Memory ] |
| FirstUserSeg | CONST | [ Module RunWrite ] |
| FirstUserSeg | VAR | [ Module RunRead ] |
| FirstUserSeg | VAR | [ Module RunRead ] |
| FirstWindp | VAR | [ Module Screen ] |
| FLOPBITS | CONST | [ Module DiskIO ] |
| FlopHdArray | TYPE | [ Module DiskDef ] |
| FlopHeadPtr | TYPE | [ Module DiskDef ] |
| Floppy | CONST | [ Module IO_Unit ] |
| FLOPPYNUMBER | CONST | [ Module DiskIO ] |
| FlpUnitNumber | TYPE | [ Module VolumeSystem ] |
| FNString | TYPE | [ Module Code ] |
| Font | TYPE | [ Module Screen ] |
| FontFile | CONST | [ Module FileTypes ] |
| FontPtr | TYPE | [ Module Screen ] |
| FontSeg | CONST | [ Module Memory ] |
| FootAr | TYPE | [ Module PopUpCurs ] |
| footW | VAR | [ Module PopUpCurs ] |
| ForFile | CONST | [ Module FileTypes ] |
| format | VAR | [ Module PasReal ] |
| FracDigits | VAR | [ Module PasReal ] |
| FSBit16 | TYPE | [ Module FileDefs ] |
| FSBit32 | TYPE | [ Module FileDefs ] |
| FSBit8 | TYPE | [ Module FileDefs ] |
| FSDataEntry | TYPE | [ Module FileDefs ] |
| FSDebug | CONST | [ Module FileSystem ] |
| FSDirPrefix | VAR | [ Module FileSystem ] |
| FSOpenType | TYPE | [ Module FileDefs ] |
| FSpDB | CONST | [ Module DiskDef ] |
| FSpT | CONST | [ Module DiskDef ] |
| FSSysSearchList | VAR | [ Module FileSystem ] |
| FSVersion | CONST | [ Module FileSystem ] |
| FTPPacket | TYPE | [ Module FTPUtils ] |
| FudgeStack | CONST | [ Module Code ] |
| GetTSVersion | CONST | [ Module GetTimeStamp ] |
| gpacg | CONST | [ Module gpib ] |
| gpAuxiliaryCommands | TYPE | [ Module gpib ] |
| gpBuffer | TYPE | [ Module gpib ] |
| gpBufMax | CONST | [ Module gpib ] |
| gpBufPtr | VAR | [ Module gpib ] |
| gpBufSize | CONST | [ Module gpib ] |
| gpByte | TYPE | [ Module gpib ] |

| | | |
|---|---|---|
| gpCommandBuffer | VAR | [ Module gpib ] |
| gpCommandBuffer | VAR | [ Module gpib ] |
| gpdcl | CONST | [ Module gpib ] |
| gpDeviceAddress | TYPE | [ Module gpib ] |
| gpget | CONST | [ Module gpib ] |
| gpgtl | CONST | [ Module gpib ] |
| gpHaveAuxiliaryCommands | VAR | [ Module gpib ] |
| gpHaveDataBytes | VAR | [ Module gpib ] |
| GPIBIntMask | VAR | [ Module IOGPIB ] |
| GPIBpadConnected | TYPEP | [ Module IO_Others ] |
| GPIBStat | TYPE | [ Module IO_Unit ] |
| GPIBTabBuf | VAR | [ Module IOGPIB ] |
| GPIBTabletState | VAR | [ Module IOGPIB ] |
| GpibVersion | CONST | [ Module gpib ] |
| GPIBxFudge | CONST | [ Module IOGPIB ] |
| GPIByFudge | CONST | [ Module IOGPIB ] |
| gplag | CONST | [ Module gpib ] |
| gpllo | CONST | [ Module gpib ] |
| gpmla | CONST | [ Module gpib ] |
| gpmsa | CONST | [ Module gpib ] |
| gpmta | CONST | [ Module gpib ] |
| gpParmType | TYPE | [ Module gpib ] |
| gppBuffer | TYPE | [ Module gpib ] |
| gpppc | CONST | [ Module gpib ] |
| gpppd | CONST | [ Module gpib ] |
| gpppe | CONST | [ Module gpib ] |
| gpppu | CONST | [ Module gpib ] |
| gpRange | TYPE | [ Module gpib ] |
| gpscg | CONST | [ Module gpib ] |
| gpsdc | CONST | [ Module gpib ] |
| gpspd | CONST | [ Module gpib ] |
| gpspe | CONST | [ Module gpib ] |
| gptag | CONST | [ Module gpib ] |
| gptct | CONST | [ Module gpib ] |
| gpuag | CONST | [ Module gpib ] |
| gpunl | CONST | [ Module gpib ] |
| gpunt | CONST | [ Module gpib ] |
| HardDisk | CONST | [ Module IO_Unit ] |
| HARDNUMBER | CONST | [ Module DiskIO ] |
| HdPtr | | [ Module IODisk ] |
| HdPtr | VAR | [ Module IO_Unit ] |
| HdrPtr | VAR | [ Module DiskDef ] |
| Header | VAR | [ Module RunRead ] |
| Header | TYPE | [ Module DiskIO ] |
| HisAddr | VAR | [ Module FTPUtils ] |
| HisName | VAR | [ Module FTPUtils ] |
| HiVolBlock | TYPE | [ Module IO_Private ] |
| Identifier | TYPE | [ Module Stream ] |
| IdentLength | CONST | [ Module Stream ] |
| IdentTable | TYPE | [ Module Stream ] |
| IDType | TYPE | [ Module UserPass ] |
| ImpNode | TYPE | [ Module Code ] |
| in | VARW | [ Module Screen ] |
| IncludeFile | CONST | [ Module FileTypes ] |
| InCmdFile | VAR | [ Program System ] |
| indep | TYPEP | [ Module IO_Others ] |

| | | |
|---|---|---|
| INDSIZE | CONST | [ Module DiskIO ] |
| Initialized | VAR | [ Module DiskDef ] |
| InPmd | VAR | [ Program System ] |
| IntArray | TYPE | [ Module QuickSort ] |
| IntDiskKinds | TYPE | [ Module VolumeSystem ] |
| IntDiskType | VAR | [ Module DiskDef ] |
| IntType | VAR | [ Module IO_Unit ] |
| IntUnitNumber | TYPE | [ Module VolumeSystem ] |
| IO24MByte | VAR | [ Module IO_Unit ] |
| IOBuffer | TYPE | [ Module IO_Unit ] |
| IOBufPtr | TYPE | [ Module IO_Unit ] |
| IOCommands | TYPE | [ Module IO_Unit ] |
| IOEABN | CONST | [ Module IOErrors ] |
| IOEADR | CONST | [ Module IOErrors ] |
| IOEBAE | CONST | [ Module IOErrors ] |
| IOEBSE | CONST | [ Module IOErrors ] |
| IOEBUN | CONST | [ Module IOErrors ] |
| IOECBF | CONST | [ Module IOErrors ] |
| IOECDI | CONST | [ Module IOErrors ] |
| IOECMM | CONST | [ Module IOErrors ] |
| IOECOR | CONST | [ Module IOErrors ] |
| IOEDAC | CONST | [ Module IOErrors ] |
| IOEDNI | CONST | [ Module IOErrors ] |
| IOEDNR | CONST | [ Module IOErrors ] |
| IOEDNW | CONST | [ Module IOErrors ] |
| IOEDRS | CONST | [ Module IOErrors ] |
| IOEFirstError | CONST | [ Module IOErrors ] |
| IOEFirstError | CONST | [ Module IOErrors ] |
| IOEFLT | CONST | [ Module IOErrors ] |
| IOEFRS | CONST | [ Module IOErrors ] |
| IOEILC | CONST | [ Module IOErrors ] |
| IOEIOB | CONST | [ Module IOErrors ] |
| IOEIOC | CONST | [ Module IOErrors ] |
| IOELastError | CONST | [ Module IOErrors ] |
| IOELHB | CONST | [ Module IOErrors ] |
| IOELHC | CONST | [ Module IOErrors ] |
| IOELHE | CONST | [ Module IOErrors ] |
| IOELHS | CONST | [ Module IOErrors ] |
| IOEMDA | CONST | [ Module IOErrors ] |
| IOEMHA | CONST | [ Module IOErrors ] |
| IOENBD | CONST | [ Module IOErrors ] |
| IOENCD | CONST | [ Module IOErrors ] |
| IOENHP | CONST | [ Module IOErrors ] |
| IOENOC | CONST | [ Module IOErrors ] |
| IOEOVR | CONST | [ Module IOErrors ] |
| IOEPHC | CONST | [ Module IOErrors ] |
| IOEPTL | CONST | [ Module IOErrors ] |
| IOERDI | CONST | [ Module IOErrors ] |
| IOESKE | CONST | [ Module IOErrors ] |
| IOESME | CONST | [ Module IOErrors ] |
| IOESNF | CONST | [ Module IOErrors ] |
| IOESOR | CONST | [ Module IOErrors ] |
| IOETO | CONST | [ Module IOErrors ] |
| IOETIM | CONST | [ Module IOErrors ] |
| IOEUDE | CONST | [ Module IOErrors ] |
| IOEUEF | CONST | [ Module IOErrors ] |

| | | |
|---|---|---|
| IOEWRF | CONST | [ Module IOErrors ] |
| IOHeader | TYPE | [ Module IO_Unit ] |
| IOHeadPtr | TYPE | [ Module IO_Unit ] |
| IOInProgress | VAR | [ Module IO_Unit ] |
| IOIntrTypes | TYPE | [ Module IO_Unit ] |
| IOPtrKludge | TYPE | [ Module IO_Private ] |
| IOSeg | CONST | [ Module Memory ] |
| IOSegNum | VAR | [ Module IO_Private ] |
| IOSegSize | CONST | [ Module Memory ] |
| IOSegSize | CONST | [ Module Memory ] |
| IOSegSize | CONST | [ Module Memory ] |
| IOStart | CONST | [ Module IO_Unit ] |
| IOStatPtr | TYPE | [ Module IO_Unit ] |
| IOStatus | TYPE | [ Module IO_Unit ] |
| IOTime | VAR | [ Program System ] |
| IO_KeyStat | TYPE | [ Module IO_Unit ] |
| isFloppy | VAR | [ Program System ] |
| Keep1 | VAR | [ Module Virtual ] |
| Keep2 | VAR | [ Module Virtual ] |
| Keep3 | VAR | [ Module Virtual ] |
| Keep4 | VAR | [ Module Virtual ] |
| Keyboard | CONST | [ Module IO_Unit ] |
| KeyBuffer | VAR | [ Module Stream ] |
| KeyEnabled | VAR | [ Module IO_Private ] |
| KeyLength | VAR | [ Module Stream ] |
| KeyNext | VAR | [ Module Stream ] |
| Kludge | VAR | [ Module Virtual ] |
| KrizInfo | VAR | [ Module IOPointDev ] |
| KrizTabConnected | TYPEP | [ Module IO_Others ] |
| KrizTablet | VAR | [ Module IO_Others ] |
| KrizXFudge | CONST | [ Module IOPointDev ] |
| KrizYfudge | CONST | [ Module IOPointDev ] |
| KSetSLen | CONST | [ Module Screen ] |
| KTBuf | VAR | [ Module IOKeyboard ] |
| LandscapeBitHeight | CONST | [ Module Screen ] |
| LandscapeBitWidth | CONST | [ Module Screen ] |
| LandscapeWordWidth | CONST | [ Module Screen ] |
| Language | TYPE | [ Module Code ] |
| LastBlk | CONST | [ Module FileSystem ] |
| LastFileName | VAR | [ Program System ] |
| LastSeg | VAR | [ Module RunRead ] |
| LastUnit | CONST | [ Module IO_Unit ] |
| LibFile | CONST | [ Module FileTypes ] |
| LightCompiler | CONST | [ Module Lights ] |
| LightHeight | CONST | [ Module Lights ] |
| LightRecalibrate | CONST | [ Module Lights ] |
| LightScavenge | CONST | [ Module Lights ] |
| LightSpacing | CONST | [ Module Lights ] |
| LightSwap | CONST | [ Module Lights ] |
| LightUsed | CONST | [ Module Lights ] |
| LightWidth | CONST | [ Module Lights ] |
| LightY | CONST | [ Module Lights ] |
| line | VAR | [ Module PopCmdParse ] |
| LineStyle | TYPE | [ Module Screen ] |
| LoaderVersion | CONST | [ Module Loader ] |
| LoadTime | VAR | [ Program System ] |

| | | |
|---|---|---|
| LogAdr | VAR | [ Module IOGPIB ] |
| LogAdr | | [ Module IOZ80 ] |
| LogAdr | VAR | [ Module IO_Unit ] |
| LogConst | CONST | [ Program System ] |
| LS | TYPE | [ Module Screen ] |
| MainVersion | CONST | [ Program System ] |
| MAXALIAS | CONST | [ Module FTPUtils ] |
| MaxCmds | CONST | [ Module CmdParse ] |
| MaxCString | CONST | [ Module CmdParse ]. |
| MaxDataBytes | CONST | [ Module Ether10IO ] |
| MAXDISKS | CONST | [ Module AllocDisk ] |
| MAXPARTCHARS | CONST | [ Module AllocDisk ] |
| MAXPARTITIONS | CONST | [ Module AllocDisk ] |
| MaxPStringSize | CONST | [ Module PERQ_String ] |
| MaxRecv | CONST | [ Module FTPUtils ] |
| MaxSegment | CONST | [ Module Memory ] |
| MaxTotalVols | CONST | [ Module VolumeSystem ] |
| MaxUnit | CONST | [ Module IO_Unit ] |
| MaxUsers | CONST | [ Module UserPass ] |
| MaxVolID | CONST | [ Module VolumeSystem ] |
| MaxWIndx | CONST | [ Module Screen ] |
| MaxWIndx | CONST | [ Module Screen ] |
| MBootFile | CONST | [ Module FileTypes ] |
| MemoryInBlocks | VAR | [ Module Memory ] |
| MemoryVersion | CONST | [ Module Memory ] |
| MicroBinary | TYPE | [ Module ControlStore ] |
| MicroFile | CONST | [ Module FileTypes ] |
| MicroFile | TYPE | [ Module ControlStore ] |
| MicroInstruction | TYPE | [ Module ControlStore ] |
| MinDataBytes | CONST | [ Module Ether10IO ] |
| MinVolID | CONST | [ Module VolumeSystem ] |
| MltCstAddr | CONST | [ Module Ether10IO ] |
| MltCstAll | CONST | [ Module Ether10IO ] |
| MltCstGrp | CONST | [ Module Ether10IO ] |
| MltCstNone | CONST | [ Module Ether10IO ] |
| MMAddress | TYPE | [ Module Memory ] |
| MMArray | TYPE | [ Module Memory ] |
| MMBit12 | TYPE | [ Module Memory ] |
| MMBit4 | TYPE | [ Module Memory ] |
| MMBit8 | TYPE | [ Module Memory ] |
| MMBlockArray | TYPE | [ Module Memory ] |
| MMEdge | TYPE | [ Module Memory ] |
| MMExtSize | TYPE | [ Module Memory ] |
| MMFirst | VAR | [ Module Memory ] |
| MMFound); | VAR | [ Module Memory ] |
| MMFree | VAR | [ Module Memory ] |
| MMFreeNode | TYPE | [ Module Memory ] |
| MMHeap | VAR | [ Module Memory ] |
| MMHole | VAR | [ Module Memory ] |
| MMIntSize | TYPE | [ Module Memory ] |
| MMLast | VAR | [ Module Memory ] |
| MMMaxBlocks | CONST | [ Module Memory ] |
| MMMaxCount | CONST | [ Module Memory ] |
| MMMaxExtSize | CONST | [ Module Memory ] |
| MMMaxIntSize | CONST | [ Module Memory ] |
| MMNotFound | VAR | [ Module Memory ] |

| | | |
|---|---|---|
| MMPointer | TYPE | [ Module Memory ] |
| MMPosition | TYPE | [ Module Memory ] |
| MMScan10 | VAR | [ Module Memory ] |
| MMScan11 | VAR | [ Module Memory ] |
| MMScan6 | VAR | [ Module Memory ] |
| MMScan7 | VAR | [ Module Memory ] |
| MMScan8 | VAR | [ Module Memory ] |
| MMScan9 | VAR | [ Module Memory ] |
| MMState | VAR | [ Module Memory ] |
| MoveTime | VAR | [ Program System ] |
| MyAddr | VAR | [ Module FTPUtils ] |
| MyDble | TYPE | [ Module DiskIO ] |
| MyDouble | TYPE | [ Module Arith ] |
| MyName | VAR | [ Module FTPUtils ] |
| NameAr | TYPE | [ Module PopUp ] |
| NameDesc | TYPE | [ Module PopUp ] |
| newFunct | VAR | [ Module IOVideo ] |
| NextSComplemented | VAR | [ Program System ] |
| NextSOff | VAR | [ Program System ] |
| NextSSize | VAR | [ Program System ] |
| NilVolID | CONST | [ Module VolumeSystem ] |
| NumAlias | VAR | [ Module FTPUtils ] |
| number | VAR | [ Module DiskIO ] |
| NumDCBs | CONST | [ Module Ether10IO ] |
| NumDCBUsed | VAR | [ Module DiskDef ] |
| NUMTRIES | CONST | [ Module DiskIO ] |
| OffsetFile | CONST | [ Module Clock ] |
| ointDev | TYPEP | [ Module IO_Others ] |
| OldCurX | VAR | [ Module IOVideo ] |
| OldCurY | VAR | [ Module IOVideo ] |
| OldExecuteTime | VAR | [ Program System ] |
| OldIOTime | VAR | [ Program System ] |
| OldLoadTime | VAR | [ Program System ] |
| OldMoveTime | VAR | [ Program System ] |
| OldSwapTime | VAR | [ Program System ] |
| OnVolAddress | TYPE | [ Module VolumeSystem ] |
| OpenWrite | VAR | [ Module Stream ] |
| Origin | VAR | [ Module Screen ] |
| OrgX | VAR | [ Module Screen ] |
| OrgY | VAR | [ Module Screen ] |
| pArgRec | TYPE | [ Module CmdParse ] |
| PartialPathName | TYPE | [ Module FileDefs ] |
| PartitionType | TYPE | [ Module DiskIO ] |
| PartRecord | TYPE | [ Module AllocDisk ] |
| PartString | TYPE | [ Module AllocDisk ] |
| PartTable | VAR | [ Module AllocDisk ] |
| PasFile | CONST | [ Module FileTypes ] |
| PassFile | CONST | [ Module UserPass ] |
| PassType | TYPE | [ Module UserPass ] |
| Past | VAR | [ Module GetTimeStamp ] |
| PastStamp | VAR | [ Module GetTimeStamp ] |
| PathName | TYPE | [ Module FileDefs ] |
| pClockStat | TYPE | [ Module IO_Unit ] |
| pCmdList | TYPE | [ Module CmdParse ] |
| PDIBlock | TYPE | [ Module DiskDef ] |
| PDirBlk | TYPE | [ Module FileSystem ] |

| | | |
|---|---|---|
| PDiskCtrlBlock | TYPE | [ Module DiskDef ] |
| PDskCtrl | VAR | [ Module DiskDef ] |
| pEtherAdRec | TYPE | [ Module Ether10IO ] |
| pEtherBuffer | TYPE | [ Module Ether10IO ] |
| pEtherDCB | TYPE | [ Module Ether10IO ] |
| pEtherHeader | TYPE | [ Module Ether10IO ] |
| pEtherRegSave | TYPE | [ Module Ether10IO ] |
| pEtherStatus | TYPE | [ Module Ether10IO ] |
| PFileConst | CONST | [ Program System ] |
| pFNString | TYPE | [ Module Code ] |
| pFootAr | TYPE | [ Module PopUpCurs ] |
| pGPIBStat | TYPE | [ Module IO_Unit ] |
| pHiVolBlock | TYPE | [ Module IO_Private ] |
| PhyDiskID | TYPE | [ Module VolumeSystem ] |
| pImpNode | TYPE | [ Module Code ] |
| PIntArray | TYPE | [ Module QuickSort ] |
| pKeyStat | TYPE | [ Module IO_Unit ] |
| PLogHead | | [ Module DiskUtility ] |
| PLogHead | | [ Module DiskUtility ] |
| pMMArray | TYPE | [ Module Memory ] |
| pMMBlockArray | TYPE | [ Module Memory ] |
| pms255 | TYPE | [ Module PMatch ] |
| pNameAr | TYPE | [ Module PopUp ] |
| pNameDesc | TYPE | [ Module PopUp ] |
| pointAllowed | VAR | [ Program System ] |
| PointBuf | TYPE | [ Module IOPointDev ] |
| PointDev = 11; Tablet | CONST | [ Module IO_Unit ] |
| PointDevStat | TYPE | [ Module IO_Unit ] |
| PointX | VAR | [ Module IOVideo ] |
| PointY | VAR | [ Module IOVideo ] |
| popOK: boolean) | VAR | [ Module PopCmdParse ] |
| PortraitBitHeight | CONST | [ Module Screen ] |
| PortraitBitWidth | CONST | [ Module Screen ] |
| PortraitWordWidth | CONST | [ Module Screen ] |
| pPointBuf | TYPE | [ Module IOPointDev ] |
| pPointDevStat | TYPE | [ Module IO_Unit ] |
| PrintStatistics | VAR | [ Program System ] |
| ProfStr | TYPE | [ Module Profile ] |
| prompt: String; def | VAR | [ Module PopCmdParse ] |
| pRS232Stat | TYPE | [ Module IO_Unit ] |
| pSAT | TYPE | [ Module Memory ] |
| pSegBlock | TYPE | [ Module Code ] |
| pSegNode | TYPE | [ Module Code ] |
| PsgFile | CONST | [ Module FileTypes ] |
| pSIT | TYPE | [ Module Memory ] |
| PStrArray | TYPE | [ Module QuickSort ] |
| pStreamBuffer | TYPE | [ Module Stream ] |
| PString | TYPE | [ Module PERQ_String ] |
| pSwitchRec | TYPE | [ Module CmdParse ] |
| pSysNames | TYPE | [ Module Memory ] |
| ptr | | [ Module FileAccess ] |
| ptr | | [ Module FileAccess ] |
| PtrDCA | VAR | [ Module DiskDef ] |
| ptrDiskBuffer | TYPE | [ Module DiskIO ] |
| PtrDskCtrlArray | TYPE | [ Module DiskDef ] |
| ptrFSDataEntry | TYPE | [ Module FileDefs ] |

| | | |
|---|---|---|
| ptrHeader | TYPE | [ Module DiskIO ] |
| ptrScanRecord | TYPE | [ Module FileUtils ] |
| ptrSearchList | TYPE | [ Module FileSystem ] |
| PtrVBuf | VAR | [ Module DiskDef ] |
| PtrVHBuf | VAR | [ Module DiskDef ] |
| ptrVolBuffer | TYPE | [ Module VolumeSystem ] |
| ptrVolHeaderBuffer | TYPE | [ Module VolumeSystem ] |
| pUDeviceTable | TYPE | [ Module IO_Private ] |
| pUDevTab | VAR | [ Module IO_Private ] |
| puSClkDCB | TYPE | [ Module Ether10IO ] |
| PVolHead | | [ Module DiskUtility ] |
| PVolHead | | [ Module DiskUtility ] |
| pZ80Stat | TYPE | [ Module IO_Unit ] |
| pZ_Msg | TYPE | [ Module IO_Private ] |
| QCodeVersion | CONST | [ Module Code ] |
| QVerRange | TYPE | [ Module Code ] |
| RaiseException | VAR | [ Module IO_Private ] |
| RAnd | CONST | [ Module Raster ] |
| RAndNot | CONST | [ Module Raster ] |
| Random Index | CONST | [ Module DiskIO ] |
| RasterArray | TYPE | [ Module Raster ] |
| RasterPtr | TYPE | [ Module Raster ] |
| RealMIndefinite | CONST | [ Module RealFunctions ] |
| RealMInfinity | CONST | [ Module RealFunctions ] |
| RealMLargest | CONST | [ Module RealFunctions ] |
| RealMSmallest | CONST | [ Module RealFunctions ] |
| RealPIndefinite | CONST | [ Module RealFunctions ] |
| RealPInfinity | CONST | [ Module RealFunctions ] |
| RealPLargest | CONST | [ Module RealFunctions ] |
| RealPSmallest | CONST | [ Module RealFunctions ] |
| RealRelTablet | VAR | [ Module IO_Others ] |
| RECORDIOBITS | CONST | [ Module DiskIO ] |
| RecvsPosted | VAR | [ Module EtherInterrupt ] |
| RelFile | CONST | [ Module FileTypes ] |
| ResArray | TYPE | [ Module PopUp ] |
| ResRes | TYPE | [ Module PopUp ] |
| RFileFormat | CONST | [ Module Code ] |
| RFileName | VAR | [ Program System ] |
| RListHead | VAR | [ Module EtherInterrupt ] |
| RListTail | VAR | [ Module EtherInterrupt ] |
| RNot | CONST | [ Module Raster ] |
| ROr | CONST | [ Module Raster ] |
| ROrNot | CONST | [ Module Raster ] |
| RRpl | CONST | [ Module Raster ] |
| RS110 | CONST | [ Module IO_Unit ] |
| RS1200 | CONST | [ Module IO_Unit ] |
| RS150 | CONST | [ Module IO_Unit ] |
| RS19200 | CONST | [ Module IO_Unit ] |
| RS232Stat | TYPE | [ Module IO_Unit ] |
| RS2400 | CONST | [ Module IO_Unit ] |
| RS300 | CONST | [ Module IO_Unit ] |
| RS4800 | CONST | [ Module IO_Unit ] |
| RS600 | CONST | [ Module IO_Unit ] |
| RS9600 | CONST | [ Module IO_Unit ] |
| RSA=4;RS232Out=RSA;RS232In | CONST | [ Module IO_Unit ] |
| RSB | CONST | [ Module IO_Unit ] |

| | | |
|---|---|---|
| RSExt | CONST | [ Module IO_Unit ] |
| RSIntMask | VAR | [ Module IORS ] |
| RS_MaxBytes | CONST | [ Module IO_Unit ] |
| RS_MaxWords | CONST | [ Module IO_Unit ] |
| RS_StatusType | TYPE | [ Module IO_Unit ] |
| RS_WrtReg | TYPE | [ Module IO_Unit ] |
| RunElement | TYPE | [ Module Code ] |
| RunFile | CONST | [ Module FileTypes ] |
| RunFileType | TYPE | [ Module Code ] |
| RunInfo | TYPE | [ Module Code ] |
| RunReadVersion | CONST | [ Module RunRead ] |
| RunWriteVersion | CONST | [ Module RunWrite ] |
| RXNor | CONST | [ Module Raster ] |
| RXor | CONST | [ Module Raster ] |
| s25 | TYPE | [ Module PopUp ] |
| SAT | VAR | [ Module Memory ] |
| SATarray | TYPE | [ Module Memory ] |
| SATentry | TYPE | [ Module Memory ] |
| SATSeg | CONST | [ Module Memory ] |
| Save | VAR | [ Module IO_Private ] |
| SavedSwapId | VAR | [ Program System ] |
| SBitHeight | VAR | [ Module Screen ] |
| SbitWidth | VAR | [ Module Screen ] |
| SBootFile | CONST | [ Module FileTypes ] |
| SBottomY | VAR | [ Module IO_Private ] |
| ScanRecord | TYPE | [ Module FileUtils ] |
| ScreenLast | VAR | [ Module Virtual ] |
| ScreenOut | CONST | [ Module IO_Unit ] |
| ScreenSeg | CONST | [ Module Memory ] |
| ScreenVersion | CONST | [ Module Screen ] |
| SCurBitHeight | VAR | [ Module Screen ] |
| SCursorOn | VAR | [ Module Screen ] |
| SearchList | TYPE | [ Module FileSystem ] |
| SEARCHSIZELIST | CONST | [ Module FileSystem ] |
| SegBlock | TYPE | [ Module Code ] |
| SegFile | CONST | [ Module FileTypes ] |
| SegFileType | TYPE | [ Module Code ] |
| SegHint | TYPE | [ Module Code ] |
| SegID | TYPE | [ Module FileDefs ] |
| SegLength | CONST | [ Module Code ] |
| SegmentKind | TYPE | [ Module Memory ] |
| SegmentMobility | TYPE | [ Module Memory ] |
| SegmentNumber | TYPE | [ Module Memory ] |
| SegNode | TYPE | [ Module Code ] |
| SendsPosted | VAR | [ Module EtherInterrupt ] |
| SetStkBase | CONST | [ Module Memory ] |
| SetStkLimit | CONST | [ Module Memory ] |
| Setting | VAR | [ Module IO_Unit ] |
| SFunc | VAR | [ Module Screen ] |
| ShellConst | CONST | [ Program System ] |
| ShellCtrl | VAR | [ Program System ] |
| ShellName | VAR | [ Program System ] |
| ShouldReEnableSwapping | VAR | [ Program System ] |
| SimpleName | TYPE | [ Module FileDefs ] |
| SIsLandScape | VAR | [ Module Screen ] |
| SIT | VAR | [ Module Memory ] |

| | | |
|---|---|---|
| SITarray | TYPE | [ Module Memory ] |
| SITentry | TYPE | [ Module Memory ] |
| SITSeg | CONST | [ Module Memory ] |
| SLeftX | CONST | [ Module IO_Private ] |
| SListHead | VAR | [ Module EtherInterrupt ] |
| SMStatus | TYPE | [ Module DiskDef ] |
| SNArray | TYPE | [ Module Code ] |
| SMaxBitHeight | VAR | [ Module Screen ] |
| SNumTitleChars | VAR | [ Module Screen ] |
| softStat | VAR | [ Module DiskIO ] |
| Speech | CONST | [ Module IO_Unit ] |
| SpiceSegKind | TYPE | [ Module DiskIO ] |
| SRightX | CONST | [ Module IO_Private ] |
| ss25 | TYPE | [ Module QuickSort ] |
| SScreenP | VAR | [ Module Screen ] |
| SScreenW | CONST | [ Module Screen ] |
| SwapId | VAR | [ Module Memory ] |
| StackLeader | CONST | [ Module Code ] |
| StackPointer | VAR | [ Module EtherInterrupt ] |
| StackSegment | VAR | [ Module Memory ] |
| StanleyTablet | VAR | [ Module IOVideo ] |
| StartBlk | CONST | [ Module FileSystem ] |
| Stat: pEtherStatus; Bytes | CONST | [ Module EtherIOIO ] |
| StatBlk | VAR | [ Module IOFloppy ] |
| StatPtr | VAR | [ Module DiskDef ] |
| Status | VAR | [ Module Virtual ] |
| STitStrType | CONST | [ Module Screen ] |
| STopY | CONST | [ Module IO_Private ] |
| StrArray | TYPE | [ Module QuickSort ] |
| StreamBuffer | TYPE | [ Module Stream ] |
| StreamSegment | VAR | [ Module Stream ] |
| StreamVersion | CONST | [ Module Stream ] |
| StrVersion | VAR | [ Program System ] |
| StsPtr | VAR | [ Module IOGPIB ] |
| StsPtr | VAR | [ Module IOPointDev ] |
| StsPtr | VAR | [ Module IORS ] |
| StsPtr | VAR | [ Module IO_Unit ] |
| SwapFile | CONST | [ Module FileTypes ] |
| SwapId | VAR | [ Module Memory ] |
| SwappingAllowed | VAR | [ Module Memory ] |
| SwapSId | VAR | [ Module Virtual ] |
| SwapTime | VAR | [ Program System ] |
| SwitchRec | TYPE | [ Module CmdParse ] |
| Sys9s | TYPE | [ Program System ] |
| SysBootChar | VAR | [ Program System ] |
| SysDisk | VAR | [ Program System ] |
| SysFile | CONST | [ Module FileSystem ] |
| SysNameArray | TYPE | [ Module Memory ] |
| SysNameSeg | CONST | [ Module Memory ] |
| SysSegLength | CONST | [ Module Memory ] |
| SysSegName | TYPE | [ Module Memory ] |
| SystemInitialized | VAR | [ Program System ] |
| SystemVersion | VAR | [ Program System ] |
| SysTiming | CONST | [ Program System ] |
| TabAbsX | VAR | [ Module IO_Others ] |
| TabAbsY | VAR | [ Module IO_Others ] |

| | | |
|---|---|---|
| TabBlue | VAR | [ Module IO_Others ] |
| TabCount | VAR | [ Module IOVideo ] |
| TabFinger | VAR | [ Module IO_Others ] |
| TabGreen | VAR | [ Module IO_Others ] |
| TabIgnore | CONST | [ Module IOVideo ] |
| TabLeft | TYPEP | [ Module IO_Others ] |
| TabletMode | TYPE | [ Module IO_Others ] |
| TabletType | TYPE | [ Module IO_Others ] |
| TabMiddle | TYPEP | [ Module IO_Others ] |
| TabMode | VAR | [ Module IOVideo ] |
| TabMouse | VAR | [ Module IO_Others ] |
| TabRelX | VAR | [ Module IO_Others ] |
| TabRelY | VAR | [ Module IO_Others ] |
| TabRight | TYPEP | [ Module IO_Others ] |
| TabSwitch | VAR | [ Module IO_Others ] |
| TabWhite | VAR | [ Module IO_Others ] |
| TabYellow | VAR | [ Module IO_Others ] |
| TekResult | TYPE | [ Module LoadZ80 ] |
| TempFile | CONST | [ Module FileTypes ] |
| TextFile | CONST | [ Module FileTypes ] |
| TimeBuf | VAR | [ Module IO_Private ] |
| TimeFID | VAR | [ Program System ] |
| Timer | CONST | [ Module IO_Unit ] |
| TimeReference | TYPE | [ Module GetTimeStamp ] |
| times | CONST | [ Module IOGPIB ] |
| TimeStamp | TYPE | [ Module GetTimeStamp ] |
| TimeString | TYPE | [ Module Clock ] |
| TitStrLength | CONST | [ Module Screen ] |
| TotalWidth | VAR | [ Module PasReal ] |
| TotSize: integer; Mob | | [ Module BigArea ] |
| TransKey | CONST | [ Module IO_Unit ] |
| TransMicro | TYPE | [ Module ControlStore ] |
| TransMode | TYPE | [ Module FTPUtils ] |
| TRCCAdrHigh | CONST | [ Module Ether10IO ] |
| TRCCAdrMid | CONST | [ Module Ether10IO ] |
| UDeviceTable | TYPE | [ Module IO_Private ] |
| Unit | VAR | [ Module IO_Private ] |
| UnitNumber | TYPE | [ Module DiskDef ] |
| UnitRng | TYPE | [ Module IO_Unit ] |
| UnknownFile | CONST | [ Module FileTypes ] |
| uSClkDCB | TYPE | [ Module Ether10IO ] |
| UseCmd | VAR | [ Program System ] |
| UserInt | VAR | [ Program System ] |
| UserMode | VAR | [ Program System ] |
| UserPtr | VAR | [ Program System ] |
| UserRecord | TYPE | [ Module UserPass ] |
| Users | TYPE | [ Module UserPass ] |
| UsrCmdLine | VAR | [ Program System ] |
| value | VAR | [ Module PasReal ] |
| VirtualVersion | CONST | [ Module Virtual ] |
| VolAddress | TYPE | [ Module VolumeSystem ] |
| VolBlockNumber | TYPE | [ Module VolumeSystem ] |
| VolBuffer | TYPE | [ Module VolumeSystem ] |
| VolErrorCnt | VAR | [ Module VolumeSystem ] |
| VolHeaderBuffer | TYPE | [ Module VolumeSystem ] |
| VolID | TYPE | [ Module VolumeSystem ] |

| | | |
|---|---|---|
| VolIOCommand | TYPE | [ Module VolumeSystem ] |
| VolName | TYPE | [ Module VolumeSystem ] |
| VolRangeType | TYPE | [ Module VolumeSystem ] |
| Width | VAR | [ Module Screen ] |
| WindowP | TYPE | [ Module Screen ] |
| WindowType | TYPE | [ Module Screen ] |
| WinRange | TYPE | [ Module Screen ] |
| WinTable | VAR | [ Module Screen ] |
| WordSize | VAR | [ Module Stream ] |
| WpDB | CONST | [ Module DiskDef ] |
| WpFS | CONST | [ Module DiskDef ] |
| Z80 | CONST | [ Module IO_Unit ] |
| Z80Stat | TYPE | [ Module IO_Unit ] |
| Z_CmdRegister | TYPE | [ Module IO_Private ] |
| Z_Commands | TYPE | [ Module IO_Private ] |
| Z_Data | TYPE | [ Module IO_Private ] |
| Z_DataSize | CONST | [ Module IO_Private ] |
| Z_FirstData | CONST | [ Module IO_Private ] |
| Z_IntDisabled | VAR | [ Module IO_Private ] |
| Z_MaxData | CONST | [ Module IO_Private ] |
| Z_Msg | TYPE | [ Module IO_Private ] |
| Z_MsgNotAvailable | VAR | [ Module IO_Private ] |
| Z_MsgPtrKludge | TYPE | [ Module IO_Private ] |
| Z_NoData | CONST | [ Module IO_Private ] |
| Z_Queue | TYPE | [ Module IO_Private ] |
| Z_SOM | CONST | [ Module IO_Private ] |
| #Cylinders | TYPE | [ Module DiskDef ] |
| #Heads | TYPE | [ Module DiskDef ] |
| #Sectors | TYPE | [ Module DiskDef ] |