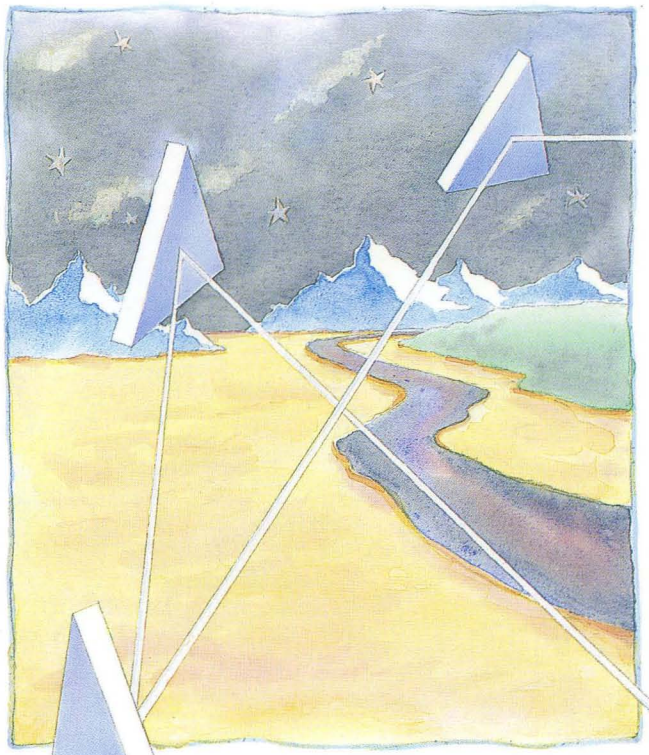


Making the Transition to SR10 Operating System Releases

apollo

Making the Transition to SR10 Operating System Releases

Order No. 011435-A02

First Printing: July 1988

Latest Printing: October 1989

10 9 8 7 6 5 4 3 2 1

Preface

Making the Transition to SR10 Operating System Releases details the changes made to the operating system at Software Release 10 (SR10). It also discusses converting an existing network of Apollo machines to SR10 and the implications of managing a mixed network of SR10 and pre-SR10 software release machines.

This document is intended mainly for programmers and system administrators who already understand the Domain/OS operating system, although skilled users may find information of value here as well.

How This Document is Organized

We've organized this manual as follows:

Chapter 1	Gives a brief high-level look at changes to the operating system at SR10.
Chapter 2	Discusses these changes in more detail, especially as they affect the general operating system environment.

Chapter 3	Describes the implications of SR10 changes in the context of system administration. It also describes issues and procedures for managing a mixed network.
Chapter 4	Discusses the SR10 changes as they affect the programming environment.
Appendix A	Provides an overview of the SR10 protection model and how it interacts with UNIX protections and pre-SR10 Domain protection.

An Index follows Appendix A.

Related Manuals

the *Domain Documentation Master Index* (011242) for a complete list of related documents. For detailed information on system administration, see the following environment-specific manuals:

Managing Aegis System Software (010852)

Managing BSD System Software (010853)

Managing SysV System Software (010851)

Managing Domain/OS and Domain Routing in an Internet (005694)

Making the Transition to SR10 TCP/IP (011717)

Printing in the Aegis Environment (011774)

Managing NCS Software (011895)

Using Your Aegis Environment (011021)

Using Your BSD Environment (011020)

Using Your SysV Environment (011022)

For information on installing SR10 software, see *Installing Software with Apollo's Release and Installation Tools* (008860).

You can order Apollo documentation by calling **1-800-225-5290**. If you are calling from outside the U.S., dial **(508) 256-6600** and ask for **Apollo Direct Channel**.

Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. To make it easy for you to communicate with us, we provide the Apollo Product Reporting (APR) system for comments related to hardware, software, and documentation. By using this formal channel, you make it easy for us to respond to your comments.

You can get more information about how to submit an APR by consulting the appropriate Command Reference manual for your environment (Aegis, BSD, or SysV). Refer to the **mkapr** (make apollo product report) shell command description. You can view the same description online by typing:

$ **man mkapr** (in the SysV environment)

% **man mkapr** (in the BSD environment)

$ **help mkapr** (in the Aegis environment)

Alternatively, you may use the Reader's Response Form at the back of this manual to submit comments about the manual.

Documentation Conventions

This manual uses the following symbolic conventions:

commands and keywords Bold words or characters in formats and command descriptions represent commands or keywords that you must use literally. Bold words in text indicate the first use of a new term. File and pathnames are also in bold.

user-supplied values	*Italic words or characters in formats and command descriptions represent values that you must supply.*
`output`	`Information that the system displays appears in this typeface. Typewriter font is also used for example program text.`
CTRL/	The notation CTRL/ followed by the name of a key indicates a control character sequence. Hold down <CTRL> while you press the key.
• • •	Vertical ellipsis points mean that irrelevant parts of a figure or example have been omitted.
———— ⊞ ————	This symbol indicates the end of a chapter.

Contents

Chapter 1 Overview of Changes

Chapter 2 Changes to the Operating System at SR10

Chapter 3 Implications for System Administrators

Chapter 4 Implications for Programmers

Appendix A Protection and ACLs

Chapter 1

Overview of Changes

Software Release 10 (SR10) of the Domain operating system, Domain/OS, provides major benefits to Apollo's customers in the form of an extended Aegis environment and enhanced support in the Domain kernel for UNIX functionality and better performance. The Domain/IX product is obsolete, and the system software now comprises one operating system and three environments.

The three SR10 operating environments can run independently of each other or concurrently. The Aegis environment provides all the functionality of the Aegis operating system, including the SR10 extensions. The BSD and SysV environments provide users with enhanced Berkeley Software Distribution 4.3 and AT&T System V Release 3 UNIX environments, respectively. SysV is compatible with the System V Release 3 Interface Definition (SVID) for Base OS, Base Libraries, and Library Extensions.

It is important to note that each of these environments runs without relying on the presence of any other. Neither of the UNIX environments, for example, requires a **/com** directory to run properly.

SR10 requires more effort to install than previous releases. Although we expect that our customers will eventually upgrade all their nodes to SR10, we realize that not all sites will be able to upgrade to SR10 all at once. Therefore, we provide the tools and information you need to operate a mixed network of pre-SR10 and SR10 machines effectively.

This document has two functions. First, it discusses changes in the

Domain system at SR10, including changes in the programming environment, as well as the implications of those changes. We expect that this information will be of interest to many system users, although particularly to programmers and system administrators. The book is also a source of information about managing the conversion of an existing network to SR10 and about operating a mixed network. These two subjects will mainly interest system administrators.

Making the Transition to SR10 Operating System Releases is organized in four parts: an overview of the changes made in SR10, a general discussion of changes to the operating system that affect all users, and two sections discussing the implications of these changes for system administrators and programmers. In all cases, when we speak of SR9.x nodes, we are referring to nodes running an operating system release of SR9.0 or higher, including SR9.7, but not running SR10. We specify the release (for example, SR9.7) where appropriate.

In many sections, we specify other manuals you can use to obtain more information on a subject. If no specific manuals are recommended, check the *Managing System Software* books for the three environments.

Compatibility with Previous Releases

SR10 has a high degree of backward compatibility. Most pre-SR10 programs will run on SR10 system software level machines, and Aegis commands and interfaces will operate as before, with few exceptions. In fact, Apollo customers with no need to operate UNIX environments will operate substantially as they have in the past, except for those areas of Aegis where we've provided extended functionality.

We've standardized our BSD and SysV commands and interfaces to make it far simpler to port UNIX applications to Domain/OS systems.

Compatibility with SR9.7

The SR9.7 version of the operating system is the primary compatibility release for SR10. In a mixed network, we strongly urge that you upgrade as many nodes to SR9.7 as possible, before you install SR10. We also urge you to port existing applications to SR9.7 before you port them to SR10. Of the pre-SR10 releases, only an SR9.7 node can copy, read, or delete files on an SR10 node.

The /sr9.7_compatibility Directory

Several compatibility issues between SR9.7 and SR10 require that we provide new SR9.7 programs or replace existing ones in order for SR9.7 nodes to operate correctly in mixed networks. SR10 includes a directory named **/sr9.7_compatibility/sr9.7_executables** that contains these programs. All SR10 nodes, depending on the environments installed, will contain some subset of the contents of this directory.

This directory includes a README file, an SR9.7 version of HoneyDanBer **uucp** to allow **uucp** to operate correctly in mixed networks, a new version of the **crpasswd** command for SR9.7 Domain/IX, the registry conversion tool (**cvtrgy**), improved versions of some other Aegis commands, new libraries that correct bugs and provide mixed network support, and new Network Computing System (NCS) libraries and commands.

Programs in this directory must execute on SR9.7 nodes. Each site should copy new versions of as many of these SR9.7 programs as applicable to as many SR9.7 nodes in the network as need them.

Another directory named **/sr9.7_compatibility/compat_with_sr9** contains its own README, **bsd4.2** and **sys5** programming tools and libraries, as well as a version of **/com/lbr** that handles objects created between SR9.5 and SR9.7. The directory also contains a library named **/swtulib**. Some routines in pre-SR10 versions of the **swtlib** library have been removed from the SR10 version of **swtlib** because the operating system does not use them. The **swtulib** library contains these routines; if you need them, you can use **inlib** to bind in the **swtulib** routines.

Compatibility with All Pre-SR10 Releases

From an SR10 node, you can copy, read, and delete files to or from nodes running any previous software release. The only potential loss of information here is in the transformation between SR10-style protections and pre-SR10 protections, although all changes in protection result in more restrictive access rather than less. See Chapters 2 and 3 for full information about SR10 protection. From pre-SR9.7 nodes, you cannot copy files to or from an SR10 node, although you can execute some network commands and non-file operations like **bldt** and **pst**.

The default object file format at SR10 is an extended version of the AT&T System V Common Object File Format (COFF). SR9.7 nodes cannot execute COFF modules, but SR10 does support the pre-SR10 **obj**

format (Apollo object file format). You can execute obj modules on both SR10 and pre-SR10 nodes.

At SR10, it is not possible to compile a module targeted to run on pre-SR9.5 systems (that is, you can't produce SR9.2 object modules on SR10). You can do this, however, for systems with software releases between SR9.5 and SR9.7, if you use the appropriate compilers.

All pre-SR10 disk volumes (floppy and removable hard disks) can be mounted on SR10 systems. However, you cannot mount SR10-format disk volumes on any pre-SR10 system, including SR9.7. The SR10 version of the **invol** utility includes an option to generate pre-SR10 format volumes. This allows you, for example, to format floppy disks on an SR10 node and mount them on pre-SR10 systems.

You cannot use an SR9.x node as a source node for Authorized Areas for SR10 installations.

Compatibility with Layered Products

Correct installation of some layered products shipped before SR10 may require that a **/com** directory already exist. At SR10, the **/com** directory is not created automatically when only a BSD or SysV environment is installed. You may have to create a **/com** directory manually to allow installation of these layered products. The products affected by this are listed in the Release Document for SR10.

Overview of SR10 Features

This section provides an overview of the features of SR10. For detailed information about many of these subjects, see the appropriate sections in later chapters. Where a feature has no particular effect on compatibility between pre-SR10 and SR10 systems, it is not discussed further here.

We've separated the changes at SR10 into four categories, as follows. Changes to the programming environment are treated separately, in Chapter 4.

- The node environment
- The file system
- The protection model and ACLs
- The registry

Prerequisites and Cautions

Although only nodes running SR9.7 or higher can communicate with a node running SR10 system software, any SR9.x node can upgrade to the SR10 version. For a node to run SR10 successfully, it must have at least 2 MB of memory; we suggest a memory size of 3 MB to 4 MB for optimal performance of many applications.

SR10 will not run on **saul** machines; that is, on nodes with model types DN100, DN400, DN420, or DN600.

Changes to disk structures at SR10 require that you reformat a node's disk with the SR10 version of the **invol** utility before you install SR10. Therefore, you will have to back up user files on each disk, **invol** the disk, then restore the files.

When you're planning the conversion process for an internet, remember that nodes cannot boot diskless over the connection between two networks. Therefore, you should probably plan on installing SR10 on at least one node in each network in an internet.

If disk space is a consideration on your network, you can install certain subsets of SR10 environments. See *Installing Software with Apollo's Release and Installation Tools* for details.

The ns_helper Process

At SR10, certain changes to the Naming Server Helper (**ns_helper**) process require the system administrator to manipulate some of the files on pre-SR10 **ns_helper** site nodes before installing the SR10 system software on those nodes. You must perform the following tasks:

1. If you currently run the **ns_helper** process on a diskless node, use the **edns** command to delete these diskless node sites from your naming server before you install SR10 on the diskless node's partner. The **ns_helper** cannot run on diskless nodes at SR10.

2. Save your **ns_helper** databases, either by copying them to another node or by using **wbak** to copy them to media, before you install SR10 on an **ns_helper** site. Once you've installed SR10, you'll restore them to a new location before restarting the **ns_helper** process. For complete details on how to accomplish this, see the sections on installation in Chapter 3.

SR10 and Domain Internets

After you install SR10 on Domain internet router nodes, you must reassign network numbers to the router's principal and alternate networks. Refer to *Managing Domain/OS and Domain Routing in an Internet* for procedures that describe how to assign network numbers.

System Initialization

Initialize the first SR10 node in a network from removable media. Once you've done this and installed the SR10 software, the node is a source for installations across the network. We suggest you install SR10 across the network from the Authorized Area the first install creates.

Because of the changes to the protection model, you cannot use a node running pre-SR10 system software as a source area for installations.

Node and User Environment Changes

We've made significant changes to the node initialization and user log-in processes. When the system software is first installed on a network, the system administrator decides which environments will be available for subsequent installations. The default environment (BSD, SysV, or Aegis) for a node is selected at node software installation time, from the choices made available by the system administrator.

Each node, disked or diskless and regardless of environment(s) installed, has a local **/etc** directory. Some entries in this directory are links to the node's **`node_data/etc** directory. Every node also has the following directories:

/usr
/usr/apollo/bin
`node_data/system_logs
`node_data/systmp

Initialization and Login

The node initialization sequence has been changed to provide greater UNIX compatibility. Instead of the Display Manager (DM), the **init** process runs as Process 1. The **init** process reads the file **/etc/rc** (which is a link to **`node_data/etc/rc**), invokes the processes specified there, reads the **/etc/ttys** file, and invokes **/etc/dm_or_spm** (which initializes the serial lines and starts the DM). Once the DM is started, the boot startup files in **`node_data** execute as usual. See *Managing System Software* for details.

We've also made changes to the log-in sequence and the way shells process startup files like **.cshrc**.

The /usr/apollo/bin Directory

Some commands and files which did not originate with the UNIX system, but are necessary to operate in a distributed environment, are now in a new directory named **/usr/apollo/bin**. (Others that have to do with system administration have been moved to the **/etc** directory.) The **/usr/apollo/bin** directory includes some **/com** commands rewritten to have both UNIX semantics and additional functionality.

It also contains new commands that must be available in all three environments and some commands that have been moved to **/usr/apollo/bin** from **/com**. In the latter case, links in **/com** point to **/usr/apollo/bin** so that shell scripts continue to work correctly. These links in **/com** may disappear at SR11. The **/usr/apollo/bin** directory is now part of the default command search rules in all shells.

Print Environment

Changes to the print environment at SR10 include enhancements to the **prf** command to provide queue and job control, a new interface for adding drivers to **prsvr**, and a print manager that allocates and coordinates print resources in a network. SR10 also supports the UNIX line printing subsystems, **lp** and **lpr**.

Process Server Creation

For situations where the DM is not available (for example, if the X Window System is the default windowing system), we provide the **/etc/server** command, which is analogous to the DM **cps** command. The **/etc/server** command allows you to run servers on a node regardless of whether anyone is logged in.

Because we expect that most Apollo sites currently use the Display Manager, we use **cps** in examples of process creation in this document. For examples of using **/etc/server** to start processes, see the *Managing System Software* books and online help pages.

SIO Lines

We provide tty support (**getty** and **init**) for remote login over SIO lines, as well as continued support for the **siologin** and **siomonit** functions.

Handling Devices and Mounting via /dev

Both SR10 UNIX environments support handling devices in the **/dev** directory and building special files with the **mknod** command.

Process Accounting

We provide UNIX process accounting on a per-node basis with **/etc/sa** and **/etc/accton**. See the online manual pages for details.

Login Monitoring

We've added a file **/etc/login_log.conf**, which is a link to the file **`node_data/login_log.conf**. You can use this file to record certain login-related events on a node. (Logins include access via **telnet** and **rlogin**.) This file allows you to specify that attempts to log in be recorded. You can record only successful attempts, only unsuccessful attempts, or both successful and unsuccessful attempts. The types of logins that you can record are as follows:

- Logins to the node's Display Manager
- Logins to a window on the node
- Logins to the node via the Server Process Manager (**spm**)
- Logins to the node via **siologin**

For more information on how to enable and manage login monitoring, see *Managing System Software* for your particular environment.

Archiving Typed Files and Changes to rbak and wbak

We've made changes to the **rbak** and **wbak** commands to make them easier to use in a mixed pre-SR10 and SR10 network and to support SR10 file systems. Additional information on **rbak** and **wbak** is in Chapter 3 of this document.

We've also added support to **tar** and **cpio** for archiving typed files. For details, see the manual pages for these commands.

Korn Shell

We provide and support the Korn shell in both BSD and SysV.

Fonts

At SR10, the keyboard allows 8-bit character input. The system software continues to convert and display fonts in the old 7-bit format. We converted most fonts to 8-bit format and provide a tool, **/sys/dm/cvt_font**, that allows you to convert permanently any 7-bit font into the new 8-bit font format. This tool can also combine dual font files that supported 8-bit fonts in previous releases into a single SR10 8-bit file.

We also provide a tool, **/sys/dm/tr_font**, that allows you to rearrange the order of the character glyphs in the font table. See online manual pages for further information.

Other features of the new font support include the following:

- Maximum 256 possible characters in an 8-bit font
- Mono and proportionally spaced fonts
- Bounding box coordinates around character
- Underlining specification

We include support for the ISO Latin-1 character set (ISO 8859/1). We've also included utilities to convert the overloaded 7-bit national fonts to the ISO 8-bit format. These utilities reside in the **/usr/apollo/bin** directory. They are listed below. See online manual pages for details on usage.

french_to_iso	Converts overloaded French to ISO format.
german_to_iso	Converts overloaded German to ISO format.
nor.dan_to_iso	Converts overloaded Norwegian and Danish to ISO format.
swedish_to_iso	Converts overloaded Swedish to ISO format.
swiss_to_iso	Converts overloaded Swiss to ISO format.
uk_to_iso	Converts overloaded UK to ISO format.

Pre-SR10 nodes do not preserve the value of the top bit of 8-bit characters. For example, if you copy files with SR10-format 8-bit character names from an SR10 node to a pre-SR10 node and back, the names will lose the data in the eighth bit.

DPSS/Mail to UNIX Mail and uucp; UNIX Mail

SR10 provides a gateway from DPSS/Mail (a layered product) to UNIX mail and to **uucp**. A field associated with person entries in the registry allows users to specify a mail address. A new command (**edsd**) allows users to edit their entries in a directory of subscribers. The SR10 BSD and SysV environments support 4.3BSD **mail** and AT&T System V Release 3 **mail**, respectively. Both forms of mail use the **sendmail** utility as a base.

UNIX Text Processing

UNIX text processing tools in Domain/IX **sys5** have been removed from the base operating system. Improved versions are available as part of DOCUMENTER'S WORKBENCH, a layered product which is available for both BSD and SysV.

2D GMR

The 2D Graphics Metafile Resource (2D GMR) is a collection of routines that provides the ability to manipulate files of 2D picture data. SR10 does not include 2D GMR with the base operating system. It is available as a layered product. This layered 2D GMR release (Release 2.1) is not compatible with Release 1.0.

Installation Tools

At SR10, we've provided new software installation tools that allow system administrators to tailor software installations to individual environments much more easily. See *Installing Software with Apollo's Release and Installation Tools* for complete details.

Integrated TCP/IP

The TCP/IP software is now a part of the base operating system. One version operates in all environments.

For complete information about user and administrative concerns, see the document *Making the Transition to SR10 TCP/IP.*

HoneyDanBer uucp

A single version of **uucp**, known as **HoneyDanBer**, operates in both UNIX environments. HoneyDanBer **uucp** is faster and more robust than previous implementations.

Since pre-SR10 Domain/IX systems use different **uucp** software, be aware of certain incompatibilities if you are maintaining a mixed network of SR10 systems and pre-SR10 Domain/IX systems. See Chapter 3 for details.

SysV STREAMS

STREAMS is the Apollo implementation of AT&T STREAMS, part of the Network Extensions to System V Release 3. It provides a framework for developing communication services. STREAMS is available as part of SysV.

The Transport Interface (TI) provides a protocol-independent interface to the transport layer of communication services developed with STREAMS. TI is based on the transport layer services defined in the OSI Transport Service Specification.

See the *Managing System Software* books for a list of STREAMS documentation available from Apollo.

File System Changes

The operating system contains both a new file system and a new, more robust directory structure. The kernel and libraries are completely case sensitive at SR10. Component names in a pathname can be up to 255 characters long; pathnames can be up to 1023 characters long. The number of possible entries in a directory is practically infinite.

These features supply faster file access, correct behavior for UNIX programs, and support for larger directories and longer names.

At SR10, instead of supporting parallel types of links, we support a single type. All UNIX commands and system calls that previously manipulated symbolic links now perform their equivalent operations on directory symbolic links.

All directories at SR10 contain the following entries:

. (dot, the current directory)
.. (dot-dot, parent of the current directory)

In Aegis, you'll only see dot and dot-dot if you use the **-h** option to the **ld** command. In UNIX environments, use the **-a** option to **ls** to see dot and dot-dot.

The default file type at SR10 is type **unstruct**. These files have no header or streams information in them. If you have programs that expect this type of information, change them to work correctly at SR10. Pre-SR9.7 nodes do not understand this file type. The default file type on SR9.7 nodes is **uasc**.

Protection Model and ACL Changes

Protection of file system objects is based on the Access Control List (ACL). The SR10 protection model integrates ACLs and the UNIX protections (modes) more tightly than at previous releases. The ACL model implements the UNIX protection style faithfully and provides extensions, as well. Protection in the two Domain/OS UNIX environments works exactly as you would expect it to work in a UNIX environment, down to the differences in default protection inheritance between 4.3BSD and System V Release 3. A complete description of the SR10 protection model in the context of the UNIX system and pre-SR10 Domain software is in Appendix A of this book.

This means that if you are a UNIX system user already, you don't need to use ACLs unless you want the extra functionality they supply. BSD and SysV users can use the following commands to manipulate ACLs: **lsacl, cpacl, dbacl,** and **chacl**. See the online manual pages for information.

SR10 ACLs have a simplified set of access rights (**pwrxk**), including a new right **k** (keep) that allows you to prevent an object from being deleted or renamed, regardless of the permissions associated with the directory in which it resides. Each object has a set of required permissions, and you can add more layers of protection with extended ACLs.

Pre-encoded, or canned, ACLs are no longer required and are not supported. The expanded ACL inheritance mechanism allows you to specify UNIX permissions inheritance via the initial file and directory ACLs. We've altered the **edacl** command to let you add an attribute to an object to specify that the object can only be locked and mapped from the home node.

A new command, **lprotect**, allows a node user to protect the node from remote access by root processes. (Root is similar, though not equivalent, to locksmith.)

SR10 also includes a command, **import_passwd**, which allows you to import **passwd** files from other systems and merge the information with an existing **passwd** file. For further information, see the online manual page for this command and the *Managing System Software* books.

You'll also find complete information about protections at SR10 in the book *Managing System Software* for your site's environment(s).

Registry Changes

The SR10 registry is based on the Apollo Network Computing System (NCS); it comprises a server mechanism and a database of name and account information. The registry and its servers can be replicated, which is useful in large networks. By virtue of this replication, the registry data is available a much higher percentage of the time than in an environment that uses a single copy of a user account information file.

The new registry provides faster access, greater availability for updates, and easier maintenance than at previous releases.

The concept of ownership within the registry allows you to decentralize registry administration by changing the ownership of groups and organizations in the registry database.

Registry information is manipulated by a single tool, the **edrgy** command. Access to most of the functionality of the **edrgy** tool is limited to the registry administrator(s). In the UNIX environments, the **passwd**, **chfn**, and **chsh** commands also operate on the registry.

At SR10, we use the UNIX password encryption algorithm to encrypt passwords stored in the registry, although SR9.x encryption is recognized as well.

Apollo's implementation of the two UNIX environments includes the **/etc/passwd** and **/etc/group** files, which contain user account and group information, respectively. Domain/OS also adds a file **/etc/org**, which contains information on organizations. (See the discussion of the registry in the *Managing System Software* books for details.) These are typed files, generated by the registry daemon automatically, and are readable but cannot be edited directly.

Changes to GPIO Support

The following general changes occur in Domain GPIO with SR10:

- All device descriptor files (ddfs) must be rebuilt with the SR10 version of the **crddf** command.
- All ddfs are now character-special devices.

- Global entry points are now case sensitive for case-sensitive languages such as C.

- A new GPIO procedure has been added to support the acquisition of devices in a streams type manager. The procedure is **pbu_$acquire_stream**.

- A new set of system calls, the scsi_$ calls, have been added to support the use of SCSI devices.

The following GPIO commands and procedures are new or changed at SR10:

- **aqdev** (changed)
- **crddf** (changed)
- **pbu_$acquire** (changed)
- **pbu_$acquire_stream** (new)

The following sections provide an overview of changes to GPIO components that are provided as part of standard system software. Refer to online pages for complete information.

The aqdev Command

At SR10, a new program, by default, is always invoked with a new process. As a transitional feature, the **aqdev** command has been changed to accommodate drivers written prior to SR10. All programs running in the current shell after acquiring the device will run **in-process** and will have access to the loaded GPIO device. For more information on **in-process** and the SR10 process model, see Chapter 4.

A new option (**-c**) allows **aqdev** to run a command instead of invoking the shell. The format for using the **-c** option is as follows:

aqdev */dev/foo* **[-d[b]] [-c** *progname arg1 arg2 ...***]**

If **aqdev** is invoked by using the **-c** option, **aqdev** will acquire the device, run *progname*, release the device, and return to the shell.

The crddf Command

We've changed the **crddf** (create device descriptor file) command in the following ways:

- All GPIO driver procedures are now case sensitive for case-sensitive languages such as C.
- Added options support the creation of SCSI ddfs
- All library pathnames are now case sensitive for case-sensitive languages such as C.
- Library pathnames have not changed in length. The maximum size of a ddf pathname is still 64 characters.
- A ddf created by the **crddf** command is now a character-special device.
- Three new options have been added to the **crddf** command for extensible streams and UNIX **open** support.

We've added the following options to the **crddf** command:

-type type_name

This option allows you to change the type UID (Unique Identifier) of the ddf file to the type of **type_name**. To do this, **crddf** determines the major device number corresponding to the type UID of **type_name**. If a major device number does not exist for the **type_name** UID, **crddf** assigns a number. The **crddf** command also assigns a major device number to **type_name** as the ddf's major device number.

-major major_device_number

This option allows you to set the major device number of the ddf file (without reference to the type UID of **type_name**, as in the **-type** option described above). To do this, **crddf** determines the type UID corresponding to the decimal

	number that you have specified as the **major_device_number**. If the major device number does not correspond to an existing type UID, it notifies you and does not set the device number. The **crddf** command also assigns the **major_device_number** to the ddf.
-minor minor_device_number	This option allows you to set the minor device number of the ddf file to the specified decimal number.

The scrch and scrattr Commands

The **/com/scrch** command, which showed screen characteristics, has been deleted from the operating system. The **/com/scrch** entry now links to the **/usr/apollo/bin/scrattr** command, which shows screen attributes.

The fppmask Command

We've removed the **/com/fppmask** command, which controlled the conditions under which floating-point exceptions were raised, from the operating system at SR10. To set these conditions, you must now use the **fpp_** system call interface. See the online help and manual pages for these calls for further information.

The Programming Environment

Chapter 4 contains an overview of significant changes to the programming environment, as well as a discussion of these changes in the context of writing and executing programs in an SR10 Domain/OS environment.

New Concepts for Aegis Users

The tighter integration of the three environments in the Domain operating system causes a number of concepts that were formerly reserved to Domain/IX to be visible to Aegis users as well. We discuss some of these here.

UNIX Identifiers

UNIX identifiers are decimal numbers associated with each primary name and Unique Identifier (UID) in the registry. These are included for UNIX compatibility and are relevant to UNIX programs. The registry conversion procedure adds UNIX identifiers to registry database entries when you convert the registry at SR10, but if you are creating a brand new network (and therefore a new registry database), you must assign UNIX identifiers when you create entries. At SR10, legal UNIX identifiers are in the range 0-65535.

The operating system reserves certain UNIX identifiers; you'll receive an error message from **edrgy** if you attempt to assign one of these to a name.

These identifiers must be unique across an internet and within a category (person, group, or organization). They don't need to be unique across categories, however; your system can have a person, group, and organization with the same UNIX identifiers.

The /etc/passwd, /etc/group, and /etc/org Files

At SR10, each node, regardless of which environment is installed, has a directory named **/etc**. Its contents depend on which environment(s) you have installed. In all environments, **/etc** will contain, among other things, three files, **passwd**, **group**, and **org**, that provide user account information to UNIX programs. Aegis-only users may ignore these files. The **/etc** directory also contains local files and commands that pertain to system operation and administration. Some entries in **/etc** are links to **`node_data/etc**.

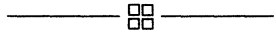
Project Lists

The project list provides a way other than accounts to allow access to file system objects (files and directories).

With BSD-based UNIX systems, a user process has a set of groups associated with it. The process group set (project list) consists of the list of groups to which a user belongs. In UNIX systems, a user's project list normally consists of the log-in group in **/etc/passwd** and any other groups to which the user belongs (from **/etc/group**). You are granted access to a file if you are a member of any group with permissions to that file.

On a Domain/OS system, the scheme is similar. A process' identity consists of a person, group, organization (**pgo**) triplet. The project list comprises the user's log-in group and all other groups in the registry that the user belongs to, assuming that the group is marked in the registry with the attribute **projlist_ok**.

The access rights allowed by the project list consist of the logical OR of the access rights allowed to each group in your project list. The project list is only created if the **PROJLIST** environment variable is true or if the **SYSTYPE** environment variable value is **BSD***xxx* (for example, BSD4.3) at login.



Chapter 2

Changes to the Operating System at SR10

In this chapter, we discuss some of the changes to the operating system in more detail. In later chapters, we'll provide information about these changes in the context of system administration and programming. We've broken this chapter into four categories:

- The node environment
- The file system
- The protection model and ACLs
- The registry

Changes to the programming environment are covered in Chapter 4.

The Node Environment

We've made substantial changes to the general node environment at SR10 in several areas, including node initialization and startup and how default environments and shells are determined.

Three User Environments

The SR10 operating system software provides three possible environments to the user: Aegis, which contains all the previous functionality of Aegis system software, as well as SR10 extensions; BSD, our implementation of the Berkeley Software Distribution 4.3 release of the UNIX operating system; and SysV, our implementation of the AT&T System V Release 3 version of the UNIX operating system.

At installation time, the system administrator chooses which environment(s) to have available on a node. If you choose to install all of the system directories and files necessary for all three environments on a node, all three environments will be available to any user of that node.

Each of these environments functions separately from the other. You can run a completely Aegis-style environment with Aegis and nothing else, either of the UNIX environments independently of Aegis, or any other configuration of one, two, or three environments.

Guaranteed Environment

In order to provide a minimum guaranteed UNIX environment so that our third-party vendors' installation scripts will run, we provide, in all operating system installations, the following SysV commands in the **/sys5.3/bin** directory:

cat	**ln**
chgrp	**ls**
chmod	**mkdir**
chown	**mv**
cp	**rm**
cpio	**rmdir**
cmp	**sed**
diff	**sort**
expr	**sum**
find	**uniq**
grep	**tar**
id	**wc**

The guaranteed UNIX shell, if no other one is available, is the SysV Bourne shell in **/etc/sys_sh**.

Differences in Project Lists

Project lists are fully implemented at SR10, which may cause some unexpected behaviors for users accustomed to the SR9.7 implementation and how it interacted with ACLs. (We assume in examples below that PROJLIST=true.)

At SR9.7, project list entries matched only ACL entries of the exact form:

```
%.proj.%.%
```

and the first exact match allowed access. If there was no exact match in the ACL entry for the project list entry, access was denied. For example, if **joeuser** had the following groups in his project list:

```
mkting
unix
development
```

and while logged in as **joeuser.mkting.r_d.%**, tried to access a file with the following ACL:

```
Acl for //node/other_user/file:
other_user.%.%.%          pgndwrx
%.development.r_d.%       pgndwrx
```

he would not gain access to the file because the ACL did not have an entry for **%.development.%.%**. **joeuser** would have had access to the file if the ACL contained entries for any or all of the following:

```
%.mkting.%.%
%.unix.%.%
%.development.%.%
```

In addition, if there had been an exactly matching entry in the file's ACL, the user would have only those access rights associated with the first exact match from the project list.

At SR10, the operating system substitutes each group in the project list into the SID and the ACL is checked for a match. The access rights associated with each ACL entry that matches a group from the project list are ORed together. That is, at SR10, the user gets all the rights associated

with ACL entries for all the groups in his project list. For example, if **joeuser** tried to access a file with the following ACL:

```
%.group1.%              r
%.othergroup.%          w
%.anothergroup.%        x
```

and his project list contained all three groups, he would have the OR of the access rights, or **rwx**. Before SR10, he would only have had the rights associated with whichever project list entry was checked first.

Shell Configuration Files

The various shells execute their configuration files as detailed in the following table:

Shell	When started by login	Always
/bin/csh	~/.login	~/.cshrc
/bin/ksh	~/.profile	~/.kshrc
/bin/sh	~/.profile	~/.shrc
/com/sh	~/user_data/sh/login	~/user_data/sh/startup

Key Definitions

At SR10, the Display Manager (DM) automatically loads key definitions at login from files in the **/sys/dm** directory. For example, nodes with the U.S. version of the Low-Profile II keyboard load the file **/sys/dm/std_keys3**.

If the SYSTYPE environment variable is set to **bsd4.3** or **sys5.3**, the DM loads the file **std_keys.unix**. Any key definitions in the **~/user_data/key_defs.xx** are then loaded, after which any other personal key definition files specified in **~/user_data/startup_dm** files are loaded.

At logout, any changes made to key definitions are saved in the appropriate **~/user_data/key_defs** file, and the appropriate **/sys/dm/std_keys** key definitions are restored.

SR10 allows you to define certain new keys. For keyboards that use the **std_keys3** keyboards only, you can combine the ESC, DEL, BS, and RETURN keys with both SHIFT and CTRL. That is, the following are now valid definable key names:

```
escs
dels
bss
crs
escc  or ^esc
delc  or ^del
bsc   or ^bs
crc   or ^cr
```

There is also a new **std_keys.basic** file that clears the ESC keys and defines the DEL keys:

```
kd del ee ke
kd dels ee ke
kd esc ke
kd escs ke
```

This provides an additional eight definable keys, at the cost of any user's existing **key_defs_8bit3** file. (See the next paragraph for a description of the **key_defs_8bit3** file.)

SR9.7 systems store the binary versions of the most recent key definitions in the **~/user_data/key_defs2** or **~/user_data/key_defs3** file. At SR10, the most recent key definitions are stored in **~/user_data/key_defs_8bit2** and **~/user_data/key_defs_8bit3**. A user's directories restored to an SR10 node may contain a **key_defs[23]** file. If it does, the DM automatically executes a program (**/sys/dm/tkd**) that creates a **key_defs_8bit** file from the **key_defs[23]** file. There is no need for users or system administrators to run the **tkd** command explicitly.

The DM loads the key definitions in the **~/user_data/keydefs_8bit3** or **~/user_data/keydefs_8bit2** file (assuming the file exists) after it has loaded all standard key definitions. Key definitions in the **~/user_data/keydefs_8bit3** or **~/user_data/keydefs_8bit2** files will override definitions set by the standard key definition file.

Standard Key Definitions

At SR10, you are still able to modify key definitions. However, some of the standard key definitions have been changed.

- The ^f5 (CTRL/F5) key can be defined to have the **compose** function by uncommenting the following line in the appropriate startup file.

  ```
  #cps /usr/apollo/bin/kbm -c f5
  ```

  The compose key enables you to enter characters that are in the upper half of the 8-bit character sequence on any keyboard. When you press the compose key followed by a 2-character sequence, the corresponding 8-bit character will be written. For example, ^F5!! corresponds to an inverted exclamation point. See the *Using Your Environment* manual for Aegis, BSD, or SysV, or the file **/usr/pub/compose** for a list of all compose keystroke combinations. The **kbm** program allows you to set other keyboard characteristics; see the online help page.

- On multinational keyboards, if you press the left ALT key before pressing any key that is marked with both ASCII and national characters, you will toggle that key between the ASCII and the national character. For example, if you are currently typing ASCII characters, pressing the ALT key makes all subsequent characters (up to another ALT) national characters. The SHIFT/ALT key combination toggles the entire keyboard between producing ASCII characters and national characters.

- There is now a single set of UNIX environment key definitions, rather than separate sets for the BSD and SysV environments. The UNIX key definitions are located in the **/sys/dm/std_keys.unix** file.

- The definitions of the READ key and m3 (right) mouse button in **std_keys.unix** have been changed to read mixed-case pathnames correctly. The previous definition quoted the entire string, including any unintentional preceding or trailing spaces, and attempted to treat it as a pathname.

- The SHELL key (l5s) now invokes the user's default log-in shell, as defined by the value of the SHELL environment variable.

- The definition of the r1s combination (SHIFT/POP) has been changed from pop to an icon toggle.
- The r2s combination (SHIFT/AGAIN) has been defined as a read-write toggle (**ro** command).
- Key definitions have been added for the numeric pad keys npa through npd.

Local /etc and `node_data/etc Directories

Each node, regardless of the environment(s) installed, has a local **/etc** directory, which contains many of the commands necessary to administer both nodes and networks. There are, in addition, some typed objects in each **/etc** directory to network-wide files like **/etc/passwd**. There are also links to the node's **`node_data** directory for specific files.

In the past, we've used a link from a node's **/etc** directory to a file in the node's **`node_data** directory as a way of implementing such files as **/etc/rc**. That is, before SR10, a node would have a link named **/etc/rc** that would resolve to a file **`node_data/etc.rc**. At SR10, we've created a **`node_data/etc** directory so that a node's **/etc/rc** link resolves to the file **`node_data/etc/rc**.

Some of the files that reside in **`node_data/etc** are listed below. All of these files may not be present, depending on which layered software packages are installed on the node; others may be present, as well.

exports
fstab
inetd.conf
mnttab
mtab
nfsstat.dat
rc
rc.local
rc.user
ttys

The /usr/apollo Directory

All SR10 systems, regardless of which environments are installed, have a **/usr/apollo** directory, which contains certain commands common to all environments, as well as Domain extensions to the UNIX environment. The directory also includes C include files for Domain system calls. It contains the following subdirectories:

/usr/apollo/bin

Wildcard expansion in Aegis environments is performed by (**/com**) commands; in UNIX environments, the shell expands wildcards. The result of this has been that executing Aegis command lines in UNIX shells (or vice versa) could cause unintended results.

To alleviate the problems of different behavior in different types of shells, and to make it possible to run either BSD or SysV without having to have a **/com** directory, we've added **/usr/apollo/bin**.

This directory contains Domain commands that are necessary in all three environments, as well as commands that extend the UNIX environments. Some of these commands previously resided in the Aegis **/com** directory, but are also useful in UNIX environments, and others are specific to the UNIX system under Domain/OS. This directory name has been added to the default path (command search rules) for all shells.

Commands that were previously available in the **/com** directory are still available in that directory. Where an identically named command is also located in **/usr/apollo**, the command in **/com** may be a link to a program in **/usr/apollo**. In other cases, particularly where the **/com** commands use derived wildcards, the version in **/com** will have different semantics from the one in **/usr/apollo**.

/usr/apollo/lib

This directory contains binaries and related files that UNIX system users do not run directly, for example, the cc compiler that the **/bin/cc** program calls. The **ftn** compiler is also in this category. The **/bin/f77** program needs to call the **ftn** compiler, which resides in **/usr/apollo/lib**.

/usr/apollo/include This directory contains Aegis system call C include files with function prototypes. Both **/sys5.3/usr/include** and **/bsd4.3/usr/include** contain links, named **apollo**, that point here. This allows you to specify **<apollo/foo.h>** in a program and requires only one copy of these include files. Versions of the Aegis system call C insert files that do not have function prototypes are located in the **/sys/ins** directory.

The Print Environment

The SR10 print environment consists of a new print architecture which adds a **print server manager** to the **prf** and **prsvr** commands.

The architecture is based on the Apollo Network Computing System, and is structured as a series of filters and drivers. Features include job query and control, and printer query and control. A sample printer driver and sample filter, along with a new manual, allow programmers to expand the system.

The Aegis print environment supports all devices we supported previously plus the Tektronix 4639d color printer. The new print environment can queue jobs from SR10 nodes to SR9.7 printer nodes and print jobs queued from SR9.7 nodes.

We have expanded UNIX printing by allowing the use of **/dev/lp**, which frees it of the requirement to use **prf**.

For more information on the Aegis print environment, see *Printing in the Aegis Environment*. For more information on the BSD or SysV print environments see the SysV and BSD *Managing System Software* books.

SIO Lines

Although **/dev/sio***x* and **/dev/tty***x* (where *x* is the port number) can refer to the same physical port, the system treats them differently. The state of DCD (Data Carrier Detect, pin #8 on a standard 25-pin RS-232 connector) is ignored on open for **/dev/sio***x*, but is meaningful for **/dev/tty***x*.

Ignoring the state of DCD on **/dev/tty** devices when calling **ios_$open** is possible by specifying **ios_$no_open_delay_opt** in the **ios_$open** call. For **/dev/sio** devices, the **ios_$no_delay_opt** is always implied.

Setting inprocess

Because of changes to the process model, you can no longer use **set inprocess** in UNIX shells as you were able to do using Domain/IX. You can, however, use **export inprocess** in Aegis shells, as follows:

```
export inprocess; inprocess := true
```

See Chapter 4 for a discussion of the single-program-per-process model.

The /com/lopstr Command

The **/com/lopstr** command lists the open streams of a process. Before SR10, when multiple programs per process were possible, this was more meaningful than at SR10. At SR10, except when INPROCESS is true, **lopstr** only reports the standard three streams open in the **lopstr** process.

Tab Settings

The Aegis and UNIX environments understand tab settings differently. To have tabs display correctly on your screen (for example, when you use the **/bin/ls** command), issue the command

```
stty -tabs
```

either from the shell or the appropriate UNIX startup file. You can also change tab settings with the DM command **ts**.

Wildcard Expansion

In Aegis environments, wildcard expansion is performed by **/com** commands; in UNIX environments, the shell expands wildcards. If you attempt to use **/com** commands with wildcards in UNIX environments, you'll see surprising and incorrect results. If you attempt to use UNIX commands in an Aegis environment, the wildcard will not be expanded at all, since neither the UNIX command nor the Aegis shell (**/com/sh**) interprets wildcards.

At SR10, we provide two versions of several commands: one in **/com**, for use in Aegis environments, and one in the **/etc** or **/usr/apollo/bin**

directories, for use in UNIX environments. Versions in **/com** will expand wildcards; versions in **/etc** or **/usr/apollo/bin** will not.

System Initialization and Startup; Login

Before SR10, a node booted in the following way:

1. In normal mode, you typed **ex aegis** at the Mnemonic Debugger prompt; in service mode, you came up to the boot shell and typed **go**.

2. The boot shell loaded the program **/sys/env**, which first loaded the global libraries from **/lib**, then loaded and ran the Display Manager from **/sys/dm/dm** on display nodes, and the Server Process Manager from **/sys/spm/spm** on non-display nodes.

3. The DM executed the file **`node_data/startup.***type*, where *type* was the appropriate monitor type (or SPM executed **`node_data/startup.spm**).

4. The DM displayed the log-in prompt and the user logged in.

At SR10, a node boots in the following way:

1. In normal mode, you type **ex domain_os** at the Mnemonic Debugger prompt (**ex aegis** will also work). In service mode, you come up to the boot shell and type **go**.

2. The boot shell loads the program **/sys/env**, which first loads the global libraries from **/lib**, using the information in the **/etc/sys.conf** file, then loads and runs the program **/etc/init**.

3. The **init** process runs in two phases. First, it runs the Bourne shell script in **/etc/rc**, which is a link to **`node_data/etc/rc**. The shell that runs this script is in **/etc/sys_sh**.

4. In its second phase, **init** reads the file **/etc/ttys**, which is a link to **`node_data/etc/ttys**. This file describes what processes to start up for the console (display) and the SIO lines. (The **siomonit** and **siologin** commands are still available for configuring a node's SIO lines, but the use of **getty** and the **/etc/ttys** file is simpler and much preferred. In the entry for the console,

/etc/ttys specifies a program called **/etc/dm_or_spm**, which **init** now starts.

5. The **/etc/dm_or_spm** program executes **/sys/dm/dm** on a node with a display, or **/sys/spm/spm** otherwise. Then the DM (or SPM) performs the rest of the startup as it happened before SR10. (Or you can specify an alternative window manager program to run in place of **/etc/dm_or_spm**.)

You will see **init** running as Process 1 on SR10 nodes.

You can revert to the pre-SR10 initialization behavior by typing **dm** or **spm** at the boot shell prompt.

The Log-In Sequence

The log-in sequence has been changed so that the startup script (**/sys/dm/startup_login.xxx**) executes the **login_sh** command, which starts up a log-in shell. It also executes the **startup_dm.***xxx* file in the **user_data** directory. (The line in the log-in startup script executing **user_data/startup_dm.***xxx* used to be commented out by default; now it is uncommented.) Also, you can no longer change your password or home directory at log-in time. That is, the **-p** and/or **-h** options to **/com/login** are no longer valid.

Default Shell Setting

Normally, **/etc/login_sh** determines what shell to start from the log-in account's default shell field in the registry. If the registry shell field is empty, the default shell is determined by the setting of the ENVIRONMENT environment variable in **/etc/environ**, either **aegis**, **bsd**, or **sysv**. The settings in **/etc/environ** are determined at installation time; do not edit this file.

The /etc/environ File

The **/etc/environ** file records the initial setting of the environment variables SYSTYPE and ENVIRONMENT for a node. The file contains comments, as well as a line to set the node's ENVIRONMENT to either **aegis**, **bsd4.3**, or **sys5.3**, and another line which sets the initial value of the SYSTYPE variable to **bsd4.3**, **sys5.3**, or blank (no SYSTYPE).

A user can override the per-node settings by creating a file in the home directory named **.environ** and resetting the variables there. The format of the **.environ** file is the same as that of the **/etc/environ** file.

The /etc/rc File

The **/etc/rc** file is a shell script that starts servers and performs other initialization tasks on a node. The SYSTYPE recorded in the **/etc/environ** file determines the environment in which **rc** actually runs. (Because **init** runs as a **bsd4.3** process, **/etc/rc** runs in a **bsd4.3** environment by default, that is, if no SYSTYPE is set in **/etc/environ**.

The file executes as a root process, so servers started from **/etc/rc** run as **root.wheel**. We suggest that you start all your servers from this file, if they run correctly as root. However, since servers started from **/etc/rc** will run as **root.wheel**, the **/etc/rc** file must be protected from editing by users other than **root**. Without being **root**, the node user can control whether servers are started, by creating or deleting files in the **`node_data/daemons** directory. If no file with the server's name exists in **`node_data/daemons**, that server will not be started by **init**, even if there is an entry in the **/etc/rc** file.

The /etc/rc.user File

This file is a link to **`node_data/etc/rc.user**. Since it executes as **user.server.none**, you can use it to start up servers that do not need to run as root, like **netman**. This file is executed by **/etc/rc**. It directly replaces that portion of the **`node_data/startup.type** file that performed server startup.

The /etc/rc.local File

The **/etc/rc.local** file is invoked by **/etc/rc**. It starts TCP/IP and its related servers.

Establishing a Default SYSTYPE

The node's default SYSTYPE is controlled by the **/etc/environ** file. Valid settings for SYSTYPE are **bsd4.3**, **sys5.3**, or [blank]. This setting is also made during the installation process. Although, in the past, node startup files have set the SYSTYPE variable, users should no longer set this variable in that way. Display Manager (DM) startup files should only be used to set DM windows.

The Mail System

A new field associated with a person's name in the registry allows the user to specify which mailer delivers that user's mail. The **edsd** command edits this field in the registry. This feature is not usable until the SR10 registry is the primary (that is, writable) registry on the network.

A UNIX mail gateway is now supplied with DPSS/Mail, which allows users to receive messages with one type of mail that were sent on another. To enable this facility, the system administrator sets the mailer field in the registry, with **edsd**, to addresses of the form *user@unix* (for DPSS/Mail to UNIX mail) and *user@dpss* (for UNIX mail to DPSS/Mail).

The **sendmail** configuration file must be changed to identify the mailer for the DPSS and UNIX gateways. An entry similar to the following should be used:

```
Mdpss, P=/usr/lib/mailer/post_dpss,
F=FDhum, S=10, R=20, A=post_dpss $u
Munix, P=/bin/mail, F=FDhum, S=10, R=20, A=mail -d   $u
```

All DPSS messages can be delivered through **sendmail**, if you wish. This option can be configured on a system-wide basis by setting the **sm** option in the configuration file **/mail/$config/ver.17**. SysV **mail** has been changed to use **sendmail** as a delivery mechanism. SysV **mail** users must have **sendmail** and **sendmail.cf** on their node. See *Managing System Software* for both UNIX environments for details of **sendmail**.

The File System

General changes to the file system include case sensitivity, higher limits on the length of both pathnames and components of pathnames, differences in links, differences in protections, the addition of two new entries to every directory, and virtually unlimited directory sizes.

At SR10, each file system object has an owner. Owner information, including UNIX identifiers, is stored with the object in the file system.

Case Sensitivity

At SR10, the system's kernel and libraries are case sensitive in handling pathnames. Any program or shell script that assumes that the system ignores case in names will not have the expected results.

The **edrgy** tool allows you to create only lowercase names in the registry database. User names are mapped to lowercase at log-in time (with **/com/login**). (The **/bin/login** command does not map names.)

Naming Server Helper

The Naming Server Helper (**ns_helper**), however, is still case insensitive; it will continue to interpret **//NODENAME** and **//nodename** the same way. We suggest you use lowercase names only for node names, since future releases may be case sensitive. For information about the implications of these changes on programs and program interfaces, see Chapter 4.

Conversion Tool: cvtname

Prior to SR10, the colon (:) was used as an escape character for the purpose of storing mixed-case names. For example, the filename **Readme** was stored as **:readme**. Domain/IX programs mapped **:r** and interpreted it as **R**. In pre-SR10 Aegis-only environments, colons used in pathnames were treated as literal characters, since Aegis was not case sensitive.

With the move to case sensitivity in all environments at SR10, colons no longer have meaning as an escape character. Thus, in converting pre-SR10 file systems to SR10, a method of selectively deciding whether a colon in a pre-SR10 pathname should be left as is or mapped to an uppercase character is necessary.

When you copy a pre-SR10 colon-mapped filename to an SR10 system (or restore to an SR10 system with **rbak**), any colon-character constructs are mapped to the appropriate uppercase equivalent automatically. To allow you to decide the treatment of colons in pathnames in a selective way, we provide the conversion tool **cvtname**.

The tool allows you to either restore colons to a pathname as literal characters or treat them as escape characters. It also allows you to convert any pathname to lowercase, uppercase, or mixed-case names. See Chapter 3 for information on running **cvtname**. The **cvtname** command resides in the **/sr9.7_compatibility/compat_with_sr9.7/com** directory.

Transition Aid: DOWNCASE Variable

We supply a transition aid for case sensitivity in the form of a per-process environment variable, DOWNCASE, which affects the way the naming server interprets pathnames. If DOWNCASE is set to **true**, then a process is said to run in downcase mode, and will be affected as described in the next paragraph; essentially, downcase mode simulates a pre-SR10 system with respect to case sensitivity and the interpretation of some special characters.

The default SR10 behavior is for DOWNCASE to be false. When DOWNCASE is false (or doesn't exist), the system is case sensitive and the naming server treats all characters literally, with certain exceptions based on usage. These exceptions and their contexts are shown in the following table:

Characters	Context
`	in `node_data only
./foo	file "foo" in this directory
../foo	file "foo" in the parent of this directory
~/foo	file "foo" in the naming directory

When a process runs in downcase mode (DOWNCASE = true), the naming server's behavior changes in two ways:

- It forces all pathnames to lowercase before it attempts to resolve them; that is, case sensitivity is turned off.
- It treats the set of characters

  ~ ` . ( )

  as syntax, and not as literal characters in a name. (This mimics the behavior of the NAMECHARS variable, which is no longer supported.)

DOWNCASE also affects interpretation of the backslash (\), dot (.), and dot-dot (..). In addition to the contexts shown above, programs running in downcase mode interprets characters as shown in the following table:

Characters	Context
\foo	file "foo" in the parent of this directory
.foo	file "foo" in this directory
..foo	file "foo" in the parent of this directory
~foo	file "foo" in the naming directory

A process running in the downcase mode cannot access mixed-case pathnames.

SR10, regardless of the setting of DOWNCASE, retains the syntactic meaning of characters in the variable link mechanism, as you can see below:

$(variable_name)

The naming server treats everything after the dollar-sign/open-parenthesis sequence, up to the closed parenthesis, as an environment variable name, and expands it to the current value of that variable. If **variable_name** is in lowercase and the operating system cannot find it, the system converts the variable name to all uppercase and tries again.

Transition Aid: The case Command

The Display Manager (DM) command **case** allows you to change the case of letters in a selected range of text. You can use this command to change uppercase letters in source code or shell scripts to lowercase. See the on-line manual pages for additional information.

Longer Names

At SR10, the maximum length of a leaf (pathname component) name is 255 characters; the maximum length of a pathname is now 1023 characters. The previous limits were 32 and 256 characters, respectively. These changes mainly affects programs that use pre-SR10 system calls. For information on new and changed system calls and interfaces, see Chapter 4. We also provide a tool that examines an object module and reports any calls that may be affected by the change in name lengths. See Chapter 4 for information.

All names returned by the naming server are null terminated.

Links

At SR10, we support a single type of symbolic link that is part of the directory structure. All UNIX commands and system calls that create, remove, or otherwise manipulate UNIX symbolic links have been changed to perform equivalent operations on directory symbolic links. This means, for example, that the command

```
ln -s foo bar
```

is functionally identical to

```
crl bar foo
```

in that both create a link named **bar** that points to an object named **foo**.

New Directory Entries

At SR10, the following appear as the first two entries in every directory:

. (dot, the current directory)
.. (dot-dot, parent of the current directory)

These directory entries cannot be deleted. Aegis users do not see these entries unless they use the new **-h** option to the **/com/ld** command. The BSD **ls -a** command displays them, and the UNIX system calls that operate on directories now return dot and dot-dot as the first two entries in a directory. The pre-SR10 Aegis directory calls do not return these entries; new naming calls do, however. See Chapter 4 for details of these calls.

The Protection Model: ACLs and Modes

If UNIX protection semantics are sufficient for your purposes, there's no need for you to learn about ACLs at all. In each of the Domain/OS UNIX environments, all the functionality associated with protection (commands, system calls) operates as it does in the UNIX type from which the environment was derived. However, since the ACL mechanism provides a more powerful protection mechanism, we make it available in the two UNIX environments via the **lsacl**, **cpacl**, **dbacl**, and **chacl** commands. Those operating in the Aegis environment will find the ACL mechanism simplified and slightly changed, as explained below. For a more exhaustive discussion of protections and how they relate to pre-SR10 Domain software, see Appendix A.

The SR10 protection model integrates two protection mechanisms, the Access Control List (ACL) model and the UNIX model, into a means of protecting both system software and user files. The SR10 model is a superset of the UNIX permissions. Changes to the protection model have been made so that we can support the UNIX protection mechanisms more gracefully, without forfeiting the fuller functionality of extended ACLs for those who have a need for them. All the features of both mechanisms are available in all three environments.

When we speak of default protection inheritance, we mean the way in which newly created file system objects receive their initial protections. As installed, the files and directories in BSD provide Berkeley default protection inheritance, and files and directories installed with SysV

provide System V default protection inheritance. Unless you use **edacl** to change the initial default ACLs, newly created objects in these environments always inherit permissions in the way their respective UNIX environments would. Directories can also be set up to allow newly created objects to inherit protection in the Aegis fashion.

We've made changes to the **acl**, **edacl**, and **salacl** commands so that they operate properly with the new ACL structures and features, and we've added commands to allow you to list, copy, and change ACLs in the two UNIX environments. See the manual pages for **lsacl**, **cpacl**, **dbacl**, and **chacl**.

Protected Subsystems

Protected subsystems have not changed at SR10.

However, since programs run by scripts are run out-of-process by default, a shell script with the following format does not work:

```
subs -up
do something which requires subsystem access
subs -down
```

You can make this work in an Aegis shell by setting the **INPROCESS** environment variable to **true**.

SID Structure

The structure of the Subject Identifier (SID) in an ACL has changed; it now consists of three fields instead of four: person, group, and organization. The node field has been removed. The major visible effect of these changes is how access rights are mapped back and forth when you're operating in a mixed network, and this is treated in Chapter 3.

Access Rights

Each file system object now has a set of required permissions for **owner**, **group**, **others** (**world**), and **organization**. The set of access rights includes the UNIX **rwx** (read, write, execute), **p** (protect), and **k** (keep) rights. Any permissions necessary beyond this required set are stored in an extended ACL, which is essentially a pre-SR10 ACL (with the access

rights limited to the aforementioned set and a three-field SID). Because the required permissions are stored with the object, no additional system overhead is involved in presenting the required permissions for an object.

In either of the UNIX environments, the protections associated with an object's owner, as reported by **stat()** and **fstat()**, are derived directly from the required entry for owner, and the protection for the object's group from the required entry for group, assuming that the SID seeking access is the owner or a member of the group. Permissions for "other" are derived from the logical OR of all permissions other than the required entries for person and group, including extended ACL entries, if any. They are reported as the appropriate combination of **rwx** for each of the three UNIX fields.

The **p** right has the same effect as before SR10; it specifies who, other than the owner, may change the object's permissions.

The **k** right is new at SR10. UNIX protections require that if one object in a directory is deletable, they all are; that is, if you have write rights in the directory, you can delete any object in the directory. The **k** right allows you to keep an object in a directory from being deleted, even if the directory is writable. In addition, the **k** right protects the name of the object. If the object has the **k** right set for some SID, that SID may not rename the object, either.

Some of the previous ACL access rights have been merged and others have been removed altogether. It is necessary to understand how pre-SR10 nodes see SR10-style ACLs, and vice versa. For complete information on the ACL rights that are no longer supported, and how ACLs map back and forth in a mixed environment, see Chapter 3.

Write-Only and Execute-Only Files

Accessing a file on the Domain system's single-level file store requires that it be mappable into the address space of the machine. At previous releases, Domain/IX automatically added read permission to any write-only or execute-only mode, to make the file mappable (that is, readable).

At SR10, in both UNIX environments, a **chmod** to a write-only (-w-) or execute-only (--x) mode results in the expected behavior. The system no longer adds the read permission.

Protection Inheritance

Another major change in the protection model at SR10 has to do with the way the permissions for a newly created object are inherited. In the past, there were two methods of inheritance: the Aegis initial file and directory ACLs, and the Domain/IX mechanism which based the protection inherited on the SID of the object's creating process.

The Domain/IX mechanism has been removed, and all protection inheritance is specified via the initial default file and directory ACLs of the containing directory. However, we've added new options to **edacl** and a new **chacl** command. These features enable you to specify access rights to be inherited and allow you to change a directory's inheritance protection mechanism. The following subsections describe the new features.

Extended edacl Features

New options to the **edacl** command enable you to alter initial default ACLs to mimic the inheritance behavior of either of the UNIX environments or of the Aegis environment. In all cases in the list below, **process** refers to the process that created the object. (Note that these options only operate on required entries, and, except for **-ignore**, only on initial default ACLs.)

-inh_all	Inherit pgo and access rights from process.
-inh_rights	Inherit only access rights from process, pgo from
-ignore	Ignore these rights when checking, but entry is on

of the required ones (pgo).

New chacl Features

Options to the **chacl** command enable you to set protection inheritance mechanism of any directory as follows:

- **-B** sets the directory to use BSD semantics.
- **-S** sets the directory to use SysV semantics.

With the **-S** option, existing ACLs are removed and the protections on the directory are determined by SysV inheritance rules: owner and group rights are inherited from the current process.

With the **-B** option, existing ACLs are removed and the protections on the directory are determined by BSD inheritance rules: owner rights are inherited from the current process, group rights from the directory.

Default Protection Inheritance Mechanisms

We emphasize again that, unless you use the **edacl** or **chacl** commands to change them, the permissions on BSD system objects, as installed, provide Berkeley-style inheritance, and the permissions on SysV provide System V-style inheritance.

The only exception to this is the behavior of **mkdir**. The **mkdir** command and the **mkdir** system call alter the way a directory inherits protections. When you use **mkdir** to create a tree now, the new directory inherits its initial file and directory ACLs from its parent directory. You can use the **/usr/apollo/bin** ACL commands (**lsacl**, **cpacl**, **dbacl**, and **chacl**) to force a particular directory to inherit protection in either of the UNIX styles (as **mkdir** previously did).

This means you must be more careful in setting up initial file and directory ACLs when you first create a directory tree, as the ACLs are propagated by UNIX **mkdir** commands.

Because of the possible combination of required entries and ignored fields, the **acl** command not only displays the entire ACL of an object but also gives the rights associated with the current process to the object.

The Local-Access-Only Attribute

As an additional measure of access protection, we support a **local-access-only** protection attribute, which specifies that an object can be locked and mapped only from the home node.

You can use the **chacl**, **lsacl**, **edacl**, and **acl** command to set and display this attribute.

Protection from Remote root Processes: lprotect

As additional protection, we've added a command, **/etc/lprotect**, that allows a node user to protect his or her node from access by a remote process running as root.

The right to run the **lprotect** command is controlled by the file **`node_data/node_owners**. The file must exist and be owned by root for **lprotect** to work. The **lprotect** command checks protection entries on that file and allows anyone with **p** rights to run **lprotect**.

A system administrator can choose whether to distribute the **`node_data/node_owners** file, in order to prevent users from using the **lprotect** command, or distribute it and allow node owners to have this control over their nodes.

This feature is useful for server nodes. It also allows the system administrator to determine the protection policy of a network and then to grant powers to individual nodes (or their owners). The **`node_data/node_owners** file also can be used to control who can **sigp** a process with a different SID on the node. At SR10, a user can only signal (using **sigp**) a process with the same SID, unless the SID of the signaling process is an owner of (has **p** rights to) **`node_data/node_owners**.

Controlling Access via spm: The spm_control File

At SR10, you can protect a node so that the Server Processor Manager (SPM) prevents unauthorized users from creating processes on or logging in to the node. If the file **`node_data/spm_control** exists on the node running **spm**, all process creation and log-in requests are validated. Only users with an SID matching an entry in the file are allowed access; all others are rejected. If the file does not exist, then all requests are allowed.

The **spm_control** file contains a list of SIDs, one per line, specifying users who are authorized. Each entry should be specified as follows:

user.group.org

where a % character in a field matches anything.

The first example shown here allows access to all users. The second allows access to all members of group **grp**.

%.%.%
%.grp.%

Obsolete Commands

Changes to the protection model have made the following commands obsolete:

fix_cache
flush_cache
addroot
sup

The Registry

The registry has changed significantly at SR10. It is a server-based and distributed system, which allows better support of very large networks and networks of different machines, and stores more account information fields.

The registry is an NCS application that consists of a database of naming and account information and a server which manages changes to the data and propagates updated information. The registry may be replicated; that is, there may be more than one copy of the database/server combination residing on different nodes in the network.

Each node communicates with a registry server node to obtain access to registry information. This is in contrast to pre-SR10 systems which directly mapped and accessed registry data files. The new server mechanism is faster and available more of the time. There is also a local registry on each node that provides account information about previous users of the node in the event that a registry server is not available.

The new registry is a distributed system, which allows better support of very large networks and networks of different machines, and stores more types of account information. Because of changes in format between pre-SR10 and SR10 registries, system administrators must manage a

conversion from the old format to the new. It is a relatively simple matter to run a mixed network using both pre-SR10 and SR10 registries, but only one set of registries can be writable at a time. The other set of registries must be regenerated periodically by the system administrator, using the tools we provide. See Chapter 3 for information on operating in a mixed network.

Registry Structure

At SR10, the registry is implemented using Apollo's Network Computing Architecture. The registry comprises a database of names and accounts, and a server that manages the database and any registry replicas. To successfully run the SR10 registry, a network must have a Global Location Broker daemon (**glbd**) running. The registry server node, and any replicas, must have a Local Location Broker daemon (**llbd**) and a registry server daemon (**rgyd**) running. For additional information on NCS and the registry, see Chapter 3, as well as *Managing System Software* for your specific environment.

Editing the Registry Database

To edit information in the registry database, you use the **edrgy** tool, which replaces the **edacct**, **cmacct**, **edppo**, and **cmppo** commands. In UNIX environments, the **passwd**, **chfn**, and **chsh** commands also operate on the registry. With **edrgy**, you can add, edit, or delete person, group, and organization names, group and organization membership lists, accounts, and policy and ownership information. Most operations in **edrgy**, with the exception of viewing and those that act on the local registry, are reserved for the registry administrator. The **edrgy** tool supports a default shell entry so that you can specify what shell users get when they log in.

In SR10, legal account names for **edrgy** must begin with a lowercase letter (a-z), and any character after the first must be either a lowercase letter (a-z), a digit (0-9), or the underscore character (_). (The **import_passwd** command enforces this restriction as well.) We did this to provide compatibility with pre-SR10 names on Apollo systems.

The **login** command no longer allows a user to change home directory or password at log-in time; that is, the **-h** and **-p** options to **login** are obsolete. You can use the Aegis **chhdir** and **chpass** commands for this purpose.

The /etc/passwd, /etc/group, and /etc/org Files

The **/etc/passwd**, **/etc/group**, and **/etc/org** files hold UNIX user account, group membership, and organization membership information, respectively. These files are required by some UNIX programs and utilities, and are automatically constructed from the registry database by the registry server (**rgyd**), and updated when the registries are updated.

In SR10, these files are typed objects with an OST type manager. None of the files is editable; to alter the information in them, you must edit the registry with **edrgy** or one of the other commands previously listed. The **vipw** command is not supplied with Domain/OS systems.

The Local Registry

Formerly, the node's local registry was found in the file **/registry/local_site**. At SR10, a node's local registry is contained in the file **/sys/registry/rgy_local**. You can edit the local registry by invoking **edrgy** with the **-l** option, on the node whose local registry you wish to edit, or on a remote node with the **-s //node** option on the **edrgy -l** command line. You can also edit a node's local registry from within **edrgy**. See online manual pages for **edrgy** for details.

Although it is necessary to convert existing registries to the SR10 format in order to run SR10 completely, it is not necessary to convert local registries since the local registry is recreated at SR10. The SR10 operating system software does not recognize the pathname **/registry/local_site** as the location of the local registry, and, in fact, the **/registry** directory does not appear on SR10 nodes.

Decentralizing Registry Administration

The SR10 registry is based on the concept of ownership. If you own a certain relation in the registry database, you can control who has the right to alter that relation. You can use this concept to partition your registry in such a way that its administration can reflect your company's organization. See *Managing System Software* for your environment for details on how to accomplish this.

The registry uses groups and organizations to separate classes of names with a common interest. The owner of the registry can delegate ownership of a organization, say, to a different person or account name, and the new owner is then responsible for administering that organization within the registry.

Required Accounts and Reserved IDs

The registry database, as provided at SR10, includes the following associations between reserved names and UNIX identifiers:

Type	Name	ID
person	root	0
person	daemon	1
person	none	12
person	user	14
person	lp	16
person	sys_person	13
person	admin	15
person	uucp	4
person	bin	3
group	wheel	0
group	daemon	1
group	none	12
group	backup	16
group	locksmith	14
group	login	15
group	mail	6
group	bin	3
group	server	18
group	sys	19
group	staff	10
group	sys_admin	17
group	sys_proj	13
org	wheel	0
org	apollo	1
org	none	12
org	sys_org	13

Access to various pieces of system software depend on these IDs. The system prevents you from changing any of them.

The registry database includes the following reserved accounts:

user.none.none
admin.none.none
bin.bin.none
daemon.none.none
lp.bin.none
root.staff.none
uucp.none.none
none.none.none
sys_user.none.none

Both system software and various subsystems like **lp** and **uucp** depend on these accounts. Don't delete them, and be careful if you need to edit them for any reason, say, to change passwords.

The SR10 registry database also includes the member relationships shown in the following table:

name	group	org
user	backup sys_admin	apollo
bin	mail	
root	bin sys	

Each name in the first column is included on the membership list for the groups or organizations specified in the second or third column. (See the *Managing System Software* books for complete information about groups, organizations, and membership lists.)

Mail System Field

Person entries in the registry now contain a field that identifies the mailer that delivers that user's mail. This field is set with the correct mail address for each mail user by the tool **edsd** (edit subscriber directory). The system administrator performs this task. This feature will operate correctly only when the SR10 registries are writable.

To allow older versions of DPSS/Mail to operate correctly with this new feature, you should specify the **rgy** command in the DPSS/Mail configuration file, **/mail/$config/ver.17**.

Obsolete Commands

Changes to the registry have made the following commands obsolete. In some cases, the function has been subsumed into **edrgy**.

lrgy
crrgy
salrgy
edacct
edppo
cmacct
cmppo
mrgrgy
adppo (unreleased)

The **crpasswd** command, which was used to build **/etc/passwd**, **/etc/group**, and **/etc/passwd.map** files from registry files, is obsolete at SR10. The **/etc/passwd.map** file, which mapped UNIX userids to Aegis UIDs at previous releases, is also obsolete.

Chapter 3

Implications for System Administrators

There are two major aspects of SR10 that affect system administration. First, unlike previous releases, SR10 cannot be installed on a disk over a previous software release. Changes to on-disk structures and to the file system, among others, require that you initialize each disk with the SR10 version of the **invol** utility before you install the SR10 software on it. This fact requires you to plan a conversion from your current SR9.x operating system software to SR10 software network-wide.

Second, although we expect SR10 will eventually be the standard software release in use at Apollo sites, we realize that all sites will not be able to convert all their nodes to SR10 at once. Therefore, the system administrator must also be concerned with operating in a mixed environment (pre-SR10 and SR10 machines on the same network or internet). We've provided information about tools and compatibility between SR10 and SR9.x in this chapter to make it possible for you to operate smoothly in a mixed network.

Before installing SR10 on the network, however, you should upgrade all nodes that will need to communicate with the SR10 nodes to the SR9.7 version of the operating system.

Installing SR10 on the First Node from Media

You must perform a media installation and initialization procedure on the first node you convert to SR10 in a network. The complete procedure for this can be found in *Installing Software with Apollo's Release and Installation Tools*. This procedure needs to be performed only once, on the first node within a network to receive SR10 software. Subsequent nodes are brought up by using a subset of that procedure and installing across the network.

We recommend that the first SR10 node be used as a source area for SR10 system software installations taking place across the network.

Installing SR10 on Other Nodes

Normally, you'll install SR10 on other nodes (after the first) by setting up the first node as an Authorized Area for the software and installing across the network. This is much simpler than installing from the media and does not require any special level of registry access. Once you've installed SR10 on a node, you need approximately 5000 blocks of free space on the node's boot volume in order to boot the node. The **salvol** utility reports the amount of free space remaining.

An overview of the steps is provided below. See *Installing Software with Apollo's Release and Installation Tools* for details.

Prerequisites for DSEE Users

If you use the Domain Software Engineering Environment (DSEE), there are several considerations you should be aware of before you install SR10. See the section in Chapter 4 entitled "SR10 and the Domain Software Engineering Environment" for details.

Back Up the Files

Because you must **invol** each disk before you install SR10 on it, you'll have to back up (and later restore) user data files that reside on the node. You can do this by copying the user directories and files to another node in the network, and then recopying to the newly installed SR10 node, or

you can use the **wbak** and **rbak** commands to back up and restore from tape. Do not back up such system directories as **/sys** or **/com**, as you will install SR10 versions of these directories.

There are considerations for using **rbak** and **wbak** in mixed networks. See the appropriate sections in this chapter.

The SR10 versions of the **rbak** and **wbak** commands automatically map any "colon-character" constructs to their uppercase equivalents. You should be aware of this if you're restoring pre-SR10 data to SR10 nodes, especially if there are literal colons in any pathnames. Use the **cvtname** conversion tool after the files have been restored to maintain literal colons in pathnames. There are also considerations having to do with the mapping of pre-SR10 ACLs to SR10 ACLs. See the section in this chapter entitled "Protection Incompatibilities."

Backing Up DSEE Files

If, instead of using **rbak** or **wbak**, you choose to use either the BSD and SysV command **cp** or the Aegis command **cpt** to copy DSEE objects to another node, you must ensure that copying preserves subsystem seals. When using **cp**, include the **-P** and **-o** options on the command line. The **-P** and **-o** options are only available with the SR10 version of **cp**. The **cpt** command preserves subsystem seals by default; do not override this behavior by issuing the command with either the **-nsubs** option or the **-dacl** option.

Suggestions for Backing Up Existing User Files

When you decide which user files to back up before you **invol** a disk, you should consider saving some or all of the files and directories listed below. This list is not exhaustive, but backing up these objects will save users considerable time in duplicating their pre-SR10 working environment on the newly installed SR10 node.

Obviously, the most important directory to save is the user's home directory. You should, however, consider saving **`node_data/startup?*** files, DM startup files from **/sys/dm**, and any printer configuration files if the node is attached to a printer. Save a copy of the top-level (/ directory) links so you'll be able to recreate them easily.

You'll probably also want to save the contents of directories like **~/com** and **/usr/local**, and any local commands you've added to **/bin** or **/com**.

(Don't make the mistake of copying entire pre-SR10 **/com** trees back onto an SR10 node just to get a few commands, though.) Also, if the node's owner has changed the icon fonts at all, save **/sys/dm/fonts/icons**.

You'll want to save the following TCP/IP administrative files. (See *Making the Transition to SR10 TCP/IP* for details.)

/etc/hosts
/etc/networks
/etc/gateways
/sys/tcp/hostmap/local.txt
/sys/tcp/hostmap/hosts.txt

Finally, you'll want to save system configuration files like **/etc/rc**, **/usr/lib/crontab**, and the **uucp** configuration files. Use the information in these files to edit the newly installed configuration file templates; in some cases, like **uucp**, configuration file formats have changed. You might also need to save passwords for third-party applications like Interleaf.

Once you've installed SR10, be sure to recreate backup lists for the user's directories.

Backing Up ns_helper Databases

If the node on which you'll be installing SR10 is an **ns_helper** database site, you must save the database files before you install SR10. The following procedure describes what you must do before installing SR10 on an **ns_helper** site node:

1. Stop the **ns_helper** process on the node, with **sigp**.

2. Copy the database files to another node; you can copy them to any pre-SR10 node.

   The following table shows the three database files, the directory where they reside on pre-SR10 file systems, and the directories where they reside in SR10 file systems.

File Name	Pre-SR10 Location	SR10 Location
ns_helper.db	/sys/node_data	/sys/ns/helper_data
ns_helper.prop	/sys/node_data	/sys/ns/helper_data
ns_helper.err_log	/sys/node_data	/sys/node_data/system_lo

3. Install SR10.

4. Copy the database files back to this node, to the new locations.

5. Start **ns_helper** again.

Using invol: Once per Disk

Installing SR10 software to a disked node includes initializing its boot disk with the SR10 version of the **invol** command. You'll find a detailed procedure for using **invol** on a disk in *Installing Software with Apollo's Release and Installation Tools.*

Install Software

We recommend that you install SR10 system software onto nodes from the Authorized Area created on the first node by the installation tool. It is possible, of course, to create multiple Authorized Areas on different nodes to facilitate installing software over large networks. You can also create a single Authorized Area that resides on more than one disk.

For complete information on installing SR10 base software, as well as about installing optional software products, see *Installing Software with Apollo's Release and Installation Tools.* The SR10 release includes completely new installation tools, which run only on SR10 nodes.

Restore

Once you've completed installation of the SR10 system software, you should restore user data to the disk. Be aware that there may be differences in how the SR9.x ACLs were restored to SR10 ACLs (see the information on ACLs in mixed networks) and that colon-character sequences in pathnames will be mapped automatically to capital letters.

Other Considerations

If the node that you've just converted to SR10 is going to be a replica registry site, you must also create a registry database and start the appropriate servers on the node. See the section entitled "Creating a Replica Registry" in this chapter.

Setting Up a Registry

When setting up a registry, you should choose as a registry site a disked node with ample memory and processes, since the node will have to run both an **llbd** process and a **rgyd** process. If your network does not have any pre-SR10 NCS applications running, you'll also have to have a **glbd** process running somewhere on the network. See later sections for information about these processes.

In order to have registry services, your first SR10 node should also be, at least temporarily, an SR10 registry site. You can move the registry site once you have other SR10 nodes up on your network. For information on how to move the master registry site, see *Managing System Software* for your particular environment.

Apollo networks with pre-SR10 registries must convert these registries from the old (pre-SR10) format to the new format. The program **/install/tools/cvtrgy** operates on an SR9.7 node to convert an SR9.x registry to an SR10-format registry; it also can convert an SR10 registry to an SR9.x registry. Only one version of the registry (SR9.x or SR10) may be writable on a network. The other version must be maintained as a read-only copy. Your site may choose to maintain its writable registry in either SR9.x or SR10 format, depending on how many nodes of each type you have in your network. However, once you have marked the SR9.x registry read-only with the **-readonly** option to **cvtrgy**, you cannot make it writable again and you cannot run **cvtrgy** in the **-from9to10** direction.

If you are a new Apollo site, you will create a registry database and add names and account information to it. In this case, no conversion is necessary.

You needn't convert local registries because SR10 creates a new local registry in the **/sys/registry** directory instead of in **/registry/local_site**, as in previous releases. The **cvtrgy** command does not operate on local registries.

If your Apollo systems share files with other UNIX systems, you must ensure that each person and group in the Apollo registry has the same user and group IDs on all of the hosts that share files.

We provide a tool called **/etc/import_passwd** that can help you to identify and resolve possible conflicts of names and IDs. Another tool called **/etc/syncids** fixes the user and group IDs stored as part of the protection information for each file and directory on an Apollo file system.

Reference documentation for **import_passwd** and **syncids** is available online and in the appropriate command reference: *SysV Command Reference* (order number 005798), *BSD Command Reference* (order number 005800), or *Aegis Command Reference* (order number 002547). For complete information about using **import_passwd**, you should refer to Chapter 4 of the *Managing System Software* manual for your environment.

Mixed networks of SR10 and SR9.x nodes must maintain two sets of registries, one in each format. SR10 nodes operate from the SR10 registries and pre-SR10 nodes from the SR9.x registries. Either set of these registries can be designated writable, but only one set can be writable at a time; the other must be updated periodically by using **cvtrgy**. In practice, until you have a substantial number of nodes running the SR10 software, you'll probably wish to have your SR10 registry be read-only. To avoid confusion and competition for resources, we suggest that you maintain SR10 format registries only on an SR10 node, and maintain your SR9.x registries on an SR9.7 node.

On a mixed network where the writable registry is in SR10 format, if you're logged in on an SR9.x node, you won't be able to change any registry information (except possibly local registry information). Instead, you'll have to make the changes from an SR10 node, then rebuild the SR9.x registry, before the changes are effective for SR9.x nodes on the network. Similar restrictions exist for SR10 nodes on networks where the writable registry is in SR9.x format.

The new subscriber directory feature of DPSS/Mail only operates when the SR10 registry is writable.

The cvtrgy Tool

The **cvtrgy** tool allows the system administrator to generate an SR10 format registry database from SR9.7 registry files; or, it generates SR9.7 registry files with data from the SR10 registry. The tool operates on SR9.7 nodes only. Whenever the conversion from 10 to 9 occurs, if a registry exists at the destination node specified in the command line, the tool quits without updating. This means that, before running **cvtrgy**, you should rename (or move) the old registry database.

The **cvtrgy** command resides in the **/install/tools** directory. In the following examples, we assume that you have the **/install/tools** directory as an entry in your PATH environment variable or as part of your **csr** (Command Search Rules). The **/install/tools** directory is not part of the default PATH and **csr** settings.

In this section, we've attempted to provide everything you need to know about **cvtrgy** to get SR10 registries running in a mixed environment. More information about **cvtrgy** is available in the online help file.

The format for invoking the **cvtrgy** shell command is as follows:

cvtrgy [-from9to10 | -from10to9] -readonly
-from //node_name -to //node_name [-nq] [-owner pgo] -first

You must be logged in as **root** or **locksmith** to run **cvtrgy**.

Operating the tool creates a read-only registry of the destination type. That is, **cvtrgy -from9to10** creates a read-only SR10 format registry, while **cvtrgy -from10to9** creates a read-only SR9.x format registry. Regardless of which direction the conversion goes (9 to 10 or 10 to 9), the SR10 registry data is updated in the replicas automatically. You must manually update the data in the SR9.x replica registries.

In order to add or change accounts and other registry data, you must edit the writable registry with the tool appropriate to the registry's format (that is, with **edrgy** on SR10, **edacct** and **edppo** on SR9.x) on a node running the same software release as the format of the writable registry. Thus, if your SR10 registries were writable, you'd have to edit them, using **edrgy**, from a node running SR10.

The **-readonly** option is only meaningful in the "9-to-10" direction, but don't use it until you decide to run your site strictly with writable SR10 registries; that is, until you are primarily an SR10 site. Once you've used

the option in a mixed network, your SR9.x registries are no longer writable, and the effect is not reversible, and you cannot run **cvtrgy** in the **-from9to10** direction again.

Performance Implications

For large networks, you should run the **cvtrgy** process at some time when it will not contend for resources with other processes. The SR9.7 node on which you run **cvtrgy** should also not run other resource-intensive services like backups, print servers, or mail delivery if you have a busy network.

How often you must run **cvtrgy** is a function of how often you modify your writable registries and how important it is that users of the read-only registries have the updated information immediately.

One side effect of running **cvtrgy -from9to10** is that you will not be able to edit the SR9.x registry while **cvtrgy** is running. When **cvtrgy -from10to9** is running, no one can log in to an SR9.x node until **cvtrgy** has completed executing and recreated the **/rgy_site** directory. We suggest, therefore, that you run the tool at a time when the registries are not heavily used.

Converting Registry Data to the SR10 Format

Before any SR10 node can have the benefit of registries, you must have the pre-SR10 registry data in the new SR10 format. If you are a new Apollo site, a new registry database will be created in the correct format. If you are not, you must convert the data.

The **cvtrgy** tool runs on SR9.7 nodes only, although it can convert registries in both directions. It actually modifies both registries, in ways we'll discuss shortly. Its primary function is to convert information in the SR9.x **ppo** and **acct** files to the SR10 registry format, but you'll also want to run it periodically in a mixed network in order to propagate changes you've made from the writable registry to the read-only one. You'll need to run **cvtrgy** to keep the two (SR9.x and SR10) registries synchronized, regardless of which registry is writable.

If there are duplicate entries in the SR9.x registry data, **cvtrgy** ignores all but the first one it encounters, and it issues a warning that it has. (You should delete duplicates at some point.)

At SR10, the registry contains canned person, group, and organization entries, as well as canned accounts. (A "canned" entry has a UID attached to it which the operating system knows about, which means that the name-UID attachment cannot be altered without affecting some operations in the system.) A list of these canned entries is shown in the following table:

Person	Group	Organization
root*	wheel	wheel
daemon	daemon	apollo*
none	none*	none*
user*	backup*	sys_org*
lp	locksmith*	
sys_person*	login*	
uucp	mail	
admin	bin	
bin	server*	
	sys	
	staff	
	sys_admin*	
	sys_proj*	

Some of these canned entries were also canned in SR9.7 registries. These are marked in the table with an asterisk (*). Where an entry is canned at SR10, but was not at SR9.7, the canned entry (name-UID association) must be propagated into the the SR9.7 registry so that the system will always associate the name with the correct UID. The **cvtrgy** tool does this as part of the registry conversion process. For example, the first time you run **cvtrgy**, it changes the **bin** entry in the SR9 registry to **bin_sr9,** and adds the SR10 **bin** canned entry to both the SR10 and SR9 registries. Where an entry was canned in both SR9.7 and SR10, **cvtrgy** does not change the SR9.7 entry.

Canned accounts at SR10 are as follows:

bin.mail.none
root.bin.none
root.sys.none
root.staff.none
daemon.none.none
none.none.none
user.none.none
lp.bin.none
admin.none.none
sys_person.none.none
uucp.daemon.none
bin.bin.none

Changes by **cvtrgy** to noncanned SR9 entries are reflected in any accounts where the person, group, or organization name appears. For example, **cvtrgy** changes the group name **staff,** to **staff_sr9** in the SR9 registry. An account previously named **root.staff.none** in the SR9 registry will then be renamed to **root.staff_sr9.none.** Since **cvtrgy** also creates the canned account **root.staff.none,** the SR9 registry has both **root.staff.none** and **root.staff_sr9.none** accounts. (Ultimately, the SR10 registry will have both, too.)

Files previously owned by **root.staff.none** in the SR9 registry are now owned by **root.staff_sr9.none,** which can cause problems in the operation of subsystems like **lp** and **uucp** on the SR9 nodes in a mixed network. If a file depends on having a specific owner in order to execute correctly (for example, files in the SysV **lp** system), you must manually change the owner from the name with the **_sr9** suffix to the appropriate name.

Since the two accounts in the registry may have different passwords, you should be aware of which account you're specifying when you log in. If you're accustomed to logging in as **root** and using the password associated with the old **root.staff.none** account, you may have to log in explicitly as **root.staff_sr9.none** for that password to be valid. You can, of course, change the default password for the SR10 **root.staff.none** to be the same as the **root.staff_sr9.none** account.

If you want to delete the **_sr9** SIDs to reduce clutter in the registry, you should first change the ACLs on any files owned by the **_sr9** SIDs so that the files are owned by active accounts. Otherwise, you may have to become **root** in order to to access the files or modify protections.

To start the first instance of a **rgyd** after you've run **cvtrgy**, log in as root and execute the following shell command:

```
/etc/server -p /etc/rgyd
```

The registry database conversion process automatically assigns the same password to all the default reserved accounts in the SR10 registry. The 8-character string **-apollo-** is the default password for all these accounts. The registry owner can change it. The account for **user** has the same password, but logging in as **user.none.none** requires no password.

The **cvtrgy** tool assigns UNIX identifiers automatically during the conversion process if you prefer. However, if your site runs Domain/IX, you will want to preserve the identifiers associated with accounts in your current (pre-SR10) **/etc/passwd** and **/etc/group** files. In normal operation, **cvtrgy** looks for the **/etc/passwd** and **/etc/group** files and assigns identifiers from them, if they exist. For this reason, you should run **cvtrgy** on an SR9.7 node that has Domain/IX installed, and either contains your master **/etc/passwd** and **/etc/group** files or has a link to them.

If **cvtrgy** doesn't find the **/etc/passwd** and **/etc/group** files and an **/etc** directory exists, it queries you before assigning new UNIX identifiers, unless the **-nq** (no query) flag is turned on, in which case **cvtrgy** exits with an error. After you've run **cvtrgy** successfully, you must run the version of **crpasswd** released with SR10 to synchronize the UNIX IDs assigned by **cvtrgy** with the ones found in the **/etc** files. This version of **crpasswd** also renames entries in the **/etc** files, if necessary. For example, it changes **bin** to **bin_sr9.** It also preserves associated information like shell field and home directory.

This version of **crpasswd** is shipped in the **/install/tools** directory, and you must copy it to the SR9.7 node where you want to run it (probably the same node where you run **cvtrgy**).

At SR10, each file system object has protection and ownership information associated with it, a UID and UNIX identifier pair for each of the object's person, group, and organization. If you change the UNIX identifier associated with the UID in the registry, the registry reassociates all names and data in the registries with the new UNIX identifier. In order to synchronize the UID and UNIX identifier in the file system if you make such a change, you must run **/etc/syncids** on all disks that might contain the old UID and UNIX identifier pair. You can also synchronize IDs, for the local registry only, from inside the **edrgy** tool.

Converting from SR9.7 to SR10

When you create the first set of SR10 registries from the SR9.x registry files, leave the SR10 registries as the read-only set for now, since the majority of your nodes and users will be working in SR9.x environments for a while yet.

You must be logged in as **root** or **locksmith** to run **cvtrgy**. That is, your SID must be **root.%.%**, or **%.locksmith.%**. Use the following shell command line to convert your SR9.x registries to an SR10 format registry database:

> **cvtrgy -from9to10 -from** *//node_name1*/**registry/rgy_site**
> **-to** *//node_name2* **-owner pgo -first**

(You only specify the **-first** option the first time you convert from SR9.7 to SR10.) You must be logged in as **root** or **locksmith** because the conversion process changes the SR9.x registries in several ways. In fact, you should control access to the node while you're running **cvtrgy**, to keep any other process from locking the SR9.x registries.

The *//node_name1* argument specifies the node on which the SR9.x registries reside; this may be the node on which you're running **cvtrgy**, or a remote one. Note that you must specify the full pathname of the SR9.x registry. The *//node_name2* argument specifies the destination SR10 node where the newly converted registry data should be placed.

The tool prompts for the name of the SR10 registry database owner. This must be a **pgo**, and an account, already in the SR9.x registries. If you omit the **-owner** flag, **cvtrgy** will prompt you for an owner, assuming that the **-nq** flag is not present. If **-nq** is on, **cvtrgy** quits with an error if you don't specify **-owner**.

If you're running **cvtrgy** with the **-first** option, you don't need either the **rgyd** or **llbd** server processes running. However, both of these servers must be running if you run **cvtrgy** without the **-first** option, that is, any time you run it after the first conversion.

Any warnings are informational only; **cvtrgy** makes no attempt to manage or "clean up" your SR9.x registries. Actual errors that stop the program require you to rerun it once you've corrected the source of the errors.

If the node on which **cvtrgy** is run has Domain/IX installed, **cvtrgy** assigns UNIX identifiers for the SR10 registry database from the node's

/etc files; if not, the tool prompts you to allow it to generate the identifiers automatically.

The newly created SR10 format registry is marked read-only.

Converting from SR10 to SR9.7

You must be logged in as **root** or **locksmith** to run **cvtrgy** in this direction, too. Use the following shell command line to convert your SR10 registries to an SR9.x format registry:

cvtrgy -from10to9 -from *//node_name1*
-to *//node_name2***/registry/rgy_site**

The **rgyd** and **llbd** servers must be running on the SR10 node.

The *//node_name1* argument specifies the node where the SR10 registry resides. The *//node_name2* argument specifies the node on which the SR9.x registries will reside; this may be the node on which you're running **cvtrgy**, or a remote one.

The **cvtrgy** tool does not overwrite the **/registry/rgy_site** file, so before you run the tool in the **10-to-9** direction, you must delete or rename the **/registry/rgy_site** file on the SR9.x destination node (*//node_name2* in the example above).

Errors that stop the program require you to rerun it, once you've corrected the source of the errors.

The SR9.x registries on this node are marked read-only. If you have multiple registry sites in your network, you must propagate the new SR9.x registries to other SR9.x registry sites in the network.

After running **cvtrgy**, you must also run the Domain/IX **crpasswd** command on an SR9.x node to update the **/etc/passwd** and **/etc/group** files. The SR10 directory **/install/tools** contains a new version of **crpasswd** which you should copy to all SR9.7 nodes that have a need to run **crpasswd**. (You can rename or replace the old version of **crpasswd**.)

Converting /etc/passwd and /etc/group Files

If you run the **cvtrgy** tool on a node where Domain/IX is installed, **cvtrgy** merges the information in these files into the SR10 registry database.

The SR10 registry database contains predefined names, some of which you may also have had in your SR9.x registries. If there are collisions between names when you convert the old registries, the pre-SR10 names are renamed with a suffix of **_sr9**. For example, the name **bin** will be renamed **bin_sr9** in both registries, and then the new entry for **bin** is added to both registries. After the conversion, both registries contain both **bin** and **bin_sr9** entries. You must also run the Domain/IX **crpasswd** command to update the **/etc/passwd** and **/etc/group** files. Use the **/sr9.7_compatibility/sr9.7_executables** version of **crpasswd**, which is shipped with SR10.

If you change UNIX IDs in the SR9.7 **/etc/passwd** or **/etc/group** files after you've already run **cvtrgy** at least once, you will want to propagate the new numbers to the SR10 registry. Use the following procedure:

1. Run **cvtrgy** with the **-invalidate_unixids** option to remove the UNIX ID information from the SR9.7 registry files.

2. Run **crpasswd** to update the password and group files.

3. Run **cvtrgy** in the **-from9to10** direction.

4. Run **/etc/syncids** on all SR10 disks.

Converting Passwords

The SR10 registry uses UNIX password encryption instead of the encryption algorithm supported in previous releases. When a user changes his or her password for the first time on an SR10 node, the system uses the new algorithm to encrypt the password. If a user does not change a password, the SR10 registry uses the SR9.x format password placed there by **cvtrgy**.

There is no need to ask users to re-encrypt their passwords (by retyping or changing them on an SR10 node), except in the following case.

If you are in a UNIX environment, and you have UNIX programs that read **/etc/passwd** and expect the password to be encrypted in the UNIX style, you will want to have all your users re-encrypt their passwords, as soon as you're ready to mark your SR10 registries writable by using the

cvtrgy -readonly option. (Remember that you use **-readonly** only once, and that its effects are not reversible.)

If this situation is relevant to your site, or you think it might become so in the future, we suggest the following strategy to force users to re-encrypt their passwords within some reasonable time after SR10 nodes become available.

Once your SR10 registry becomes writable, edit the SR10 registry database, using the **edrgy** tool, and set the **password expiration date** for all users to the previous day's date. The next time a user logs in on an SR10 node, the **login** program will prompt the user to change password.

The user can type the same password, which the registry then encrypts in the SR10 style; the user then has the same password regardless of which type of machine (SR9.x or SR10) he or she is working on. If the user types a new password, then the new password is only valid on SR10 machines, until you use the **cvtrgy** tool to propagate the changes to the SR9.x registries.

Registry Site Node Considerations

We've mentioned already that the master registry site node should be disked, and have ample memory and processes available. The **glbd** (which may run on a node other than the master registry site) runs more efficiently if it has at least 2 MB of physical memory available.

The **sr9.7_compatibility/sr9.7_executables** directory includes new SR9.7 versions of NCS software. This software includes **/lib/ddslib**, which you should copy to the **/lib** directory on the SR9.7 node, and a new **/sys/ncs** tree. You must use this new NCS software on SR9.7 nodes that run the Global Location Broker daemon, **glbd**, as well as on nodes that interact with the SR10 registry.

There must be at least one **glbd** process running on each network, and an **llbd** process must run on the same node as the **glbd** process. In addition, an **llbd** and a **rgyd** process must run on each registry site.

Another consideration is the order in which these servers are started on a node. If you already have NCS applications running on your network, you are aware of these considerations. If you are not, they are outlined below.

On the master node, start network routing services and any TCP/IP servers first, then start node servers in the following order. Start them in this order regardless of whether you start servers in the node startup file, the **/etc/rc** file, or manually. We recommend that you use the **/etc/rc** file.

1. **llbd**
2. **glbd**
3. **rgyd**
4. Other node servers

Again, the **glbd** may not necessarily be running on the master registry site node, since there is no particular advantage to having both processes running on the master registry site node.

On registry site nodes, you must adhere to the same order, leaving out the **glbd**, if appropriate. You administer the registry servers via the **rgy_admin** tool; see *Managing System Software* for your environment for details about administration.

Enabling Registry Services

The SR10 registry requires the services of several daemons (or servers) to manage and administer the SR10 registry and any replicas. An NCS component called the Location Broker enables SR10 nodes in your network to locate SR10 registries and to use their services.

Once you have a registry database, enable the registry server to provide registry services to your SR10 nodes. You obtain a registry database in one of two ways:

- If you are converting a network from SR9.7 to SR10, use **cvtrgy** to create an SR10 format registry database from the SR9.7 registry data files.

- If you are creating a new network (that is, one on which the first and all other nodes will run only SR10 software), use the **rgy_create** utility, which creates an SR10 format registry database that contains all the necessary reserved entries. (This tool

exists only in the first **/install/tools** directory created when you install from tape; network installs from that directory do not propagate **rgy_create**.)

Once you have a registry database, enable registry services by starting the registry server.

Before you start the registry server (**/etc/rgyd**), however, you must have a Global Location Broker daemon (**glbd**) running somewhere on the network, and a Local Location Broker daemon (**llbd**) running on each SR10 registry site node. For further information about the components of NCS, including administrative information, see *Managing NCS Software.*

Starting the llbd

To start the **llbd** on a node, enter the following command at the Display Manager prompt:

```
Command: cps /etc/ncs/llbd
```

You can start the **llbd** on one node from another node with the **crp** command at a shell prompt:

crp -on *//remote_node* **-cps /etc/ncs/llbd**

To start the process from the **`node_data/startup.type** file, place the following line in the file, after any lines that start network routing services or a TCP server:

```
cps /etc/ncs/llbd
```

The **/etc/rc** file will run **llbd** if the file **/etc/daemons/llbd** exists.

Starting and Administering the glbd

To start the first instance of a **glbd** on a network, enter the following command at the Display Manager prompt:

```
Command: cps /etc/ncs/glbd -create -first
```

You can start the first **glbd** on one node from another node with the **crp** command at a shell prompt:

crp -on *//remote_node* **-cps -n glbd '/etc/ncs/glbd -create -first'**

To start a replica **glbd** on a network, enter the following command at the Display Manager prompt:

```
Command: cps /etc/ncs/glbd -create -from dds://node
```

where *//node* is the node from which you want to initialize the Global Location Broker. The **glbd** starts automatically when the node is rebooted, as long as the appropriate line in **/etc/rc** is uncommented and the **/etc/daemons/glbd** exists.

A replicated GLB requires some maintenance in order to maintain consistency. See *Managing NCS Software.*

Starting the rgyd Process

To start the first instance of **rgyd** after you've run **cvtrgy**, log in as root and enter the following shell command:

/etc/server -p /etc/rgyd

The registry database conversion process automatically assigns the same password to all the default reserved accounts in the registry. The 8-character string **-apollo-** is the default password for all these accounts. The registry owner can change it. The account for **user** has the same password, but logging in as **user.none.none** requires no password.

If you create the empty file **/etc/daemons/rgyd**, the node's **/etc/rc** file, as shipped, ensures that this daemon is restarted every time the node is rebooted.

Creating a Replica Registry

To create a copy of the existing registry database, use the **rgyd** process with the **-create** flag. To create a brand new empty copy of the registry database, use the **/install/tools/rgy_create** process. See the *Managing System Software* books and the online manual page for **rgy_create** for further details.

After you have the registry database information in place on the node, you'll have to make provisions for starting the **llbd** and the **rgyd** servers on the replica registry node.

The Local Registry

A node's local registry is found in the file **/sys/registry/rgy_local** at SR10. If there is no **/sys/registry** directory on the node, the node runs without a local registry; if the **/sys/registry** directory exists, the system creates a local registry at the first login of an account other than **user.none.none**. You can use **edrgy** to change the default size and expiration settings in the local registry and to synchronize the local registry information with the master registry.

Merging Registries

If you must merge SR10 registries for any reason (for example, when you are creating an internet out of previously disjoint Apollo networks), use the **/etc/rgy_merge** command. See the online manual pages for this command, as well as the various system software and internet administration manuals.

Registry in a Single-Node Environment

In a single-node environment (or perhaps even in a two-node network), you can operate without running a registry server (or **llbd** or **glbd**), by using the limited capacity of the local registry. Briefly, you must start the registry server (**rgyd**), copy the information you need into the node's local registry via the **edrgy copy** command and stop the server. You may wish to use the **netsvc** command to take a single node out of the network and force use of its local registry. You'll then be able to run from information stored in the local registry. See the *Managing System Software* books for further details.

Cataloging Nodes

There are no changes to the **ctnode** and **uctnode** commands that catalog and uncatalog nodes in the network. Remember that node names should all be lowercase. Don't catalog nodes with mixed case names in the event that the Naming Server becomes case sensitive in a later release.

Converting Names: the cvtname Tool

The **cvtname** command allows you to choose to convert colon-mapped characters in a pathname.

You can specify, with options to **cvtname**, that the pathname(s) resulting from the conversion be either all lowercase, all uppercase, or mixed case. Another option allows you to list the conversions to take place without actually converting the names. Normally, in mixed-case mode, the tool queries you to verify that the conversions were correct; an option exists to turn this feature off.

The name conversion tool only works on an SR10 directory structure; so, before you attempt to use it, you must have initialized the node's disk and installed SR10. The **cvtname** tool will "walk" a tree for you, so you don't need to specify the names of subdirectories in a pathname, and you really only need to run the tool once.

If you don't specify **-nq**, **cvtname** presents you with both the unconverted

and converted versions of the pathname. Type **y** to convert or **n** to let the name stand. Changes take effect immediately. You can quit **cvtname** at any time by typing **q** to quit. Note that The tool will not convert some colon-character sequences in a pathname without converting them all. See the online manual pages for complete information on this tool.

Operating Mixed Networks

In a mixed network, you'll have machines with SR10 system software and machines with SR9.x system software. Any machines that do not run at least SR9.7 system software will be unable to communicate with SR10 nodes. Before you begin converting nodes to SR10, make sure that all the nodes on your network are at least at SR9.7, unless you must maintain pre-SR9.7 nodes for other reasons. Generally speaking, the major problem in mixed networks is that pre-SR9.7 nodes cannot see files and/or read directories on SR10 nodes, usually because of insufficient rights or incompatibilities in directory formats.

Copying a file (or directory) from an SR9.7 node to an SR10 (in the Aegis environment) with the **-sacl** option may limit access to the file more than you expect, since converting ACLs from SR9.7 access rights to SR10 never allows more access to an object, but can allow less.

New Versions of SR9.7 Programs

In the course of developing SR10, we altered elements in the system which caused previous versions of some SR9.7 commands to operate incorrectly or incompletely. In order to maintain complete compatibility between SR9.7 and SR10, we've rereleased any commands that were affected in this way; they are now in a directory named **/sr9.7_compatibility/sr9.7_executables**. This directory is installed as part of the standard installation of SR10; system administrators must manually copy these commands to the appropriate nodes.

The **sr9.7_compatibility/sr9.7_executables** directory includes new SR9.7 versions of NCS software. This software includes **/lib/ddslib**, which you should copy to the **/lib** directory on the SR9.7 node, and a new **/sys/ncs** tree. You must use this new NCS software on SR9.7 nodes that run the Global Location Broker daemon, **glbd**, as well as on nodes that interact with the SR10 registry.

Registry

You provide registry services to both SR9.x and SR10 nodes by maintaining both the old and the new-style registries on the network. The pre-SR10 nodes on the network use the SR9.x registries, and the SR10 registries obtain registry services from the SR10 registry servers. The pre-SR10 mechanism will continue to work as it always did, and there is no need for both types of registry to be on the same node. In fact, pre-SR10 registries should not be maintained on SR10 nodes.

Registry-Specific Cautions

Spaces and commas are not legal characters in passwords in SR9.x registries, and therefore are not present in these registries. (Pre-SR10 versions of **login** treat a space or a comma as the end of the string.) These characters are, however, legal in SR10 registries. In addition, pre-SR10 registries, because they are case insensitive, only understand lowercase passwords.

Therefore, creating a password that contains a space, comma, or uppercase letter in the writable SR10 registry makes the account associated with that password unusable on pre-SR10 machines, once you've propagated the changes to the SR9.x registries.

If your users were accustomed to typing their passwords without paying attention to case, they will have problems at SR10. Make users aware that typing with the CAPS LOCK key on is no longer a good idea.

Protection Incompatibilities in Mixed Networks

At SR10, some of the protection rights previously supported were collapsed into a smaller set of rights. This section describes changes to the access rights and their implications in mixed networks.

At SR10, passing in any of the old (supported at SR9.x but not at SR10) rights will not generate an error. But programs that examine rights bits need to be modified if they expect to see any bits from the set **gndcale** or do not expect to see **k** rights.

ACL incompatibilities in mixed networks of SR10 and SR9.x systems are limited to those that stem from the conversion of access rights. However, no conversion from SR9.x protections to SR10 ever decreases an object's protection. The following tables illustrate how ACLs map between SR9.x and SR10 nodes for access checking.

SR10 ACL Rights Interpreted by SR9

SR10 right(s)	SR9.x sees	Notes
p	pg	g right is seen if p is on
r	r	no change
w	cale	never a subset of cale
x	s	execute = search
!k	d	if not k, d
k	!d	if k, not d
	n	this right will always be on

SR9.x ACL Rights Interpreted by SR10

SR9.x right(s)	SR10 sees	Notes
p	p	no change
r	r	no change
x	x	no change
s	x	search = execute
cale	w	only if all of cale present
d	!k	if d, not k
!d	k	if not d, k
gn	rights unsupported in SR10	

It is not possible to modify any subset of **cale** rights on an SR10 directory. If, for example, an SR9.x node attempts to set **cl** rights on an SR10 directory, w rights will not be added. Only modifying all or none of the **cale** set will work correctly.

When SR9.x tries to access an SR10 directory, it can perform either all modify operations on that directory (the SR10 ACL includes **w**, which SR9.x sees as **cale**), or no modify operations (SR10 ACL has no **w**, so SR9.x sees none of **cale**). If the **k** right is set on an SR10 ACL, no **d** right is seen by an SR9.x node.

Running setuid and setgid Programs

The SR10 system software does not detect the set-user-id or set-group-id bits in the protections of objects on pre-SR10 file systems. This means that, while logged in on an SR10 node, you cannot invoke a **setuid** program that resides on a pre-SR10 node and have it run as **setuid**. This is because SR9.x systems are not able to specify the owner of a file to the SR10 operating system.

Implications for Backups

The SR10 tape format which **rbak** and **wbak** support is different from the pre-SR10 format. For this reason, any attempt to restore an SR10 format tape with a pre-SR10 version of **rbak** can cause serious errors, including (but not limited to) incorrect directory names, unusual characters, and corrupted file contents. Also for this reason, you should not create individual tapes that contain material written by both SR10 and pre-SR10 versions of **wbak**.

Protections and Backups

Restoring pre-SR10 objects to SR10 systems changes the ACLs on the object into SR10-style ACLs, with new rights. However, while rights may change in slight ways, the transformation never decreases the amount of protection that the pre-SR10 ACL provided.

SR10 rbak and wbak Commands

The **rbak** and **wbak** commands perform ACL and name conversions along the lines just described. The SR10 versions of **rbak** and **wbak** automatically map any colon-character constructions to the appropriate uppercase letter or character, regardless of whether the colon is intended to be literal. If you intend to use literal colons in pathnames, you should use the **cvtname** tool after the restore to ensure that they are reinstated.

Certain restrictions apply when you're using **rbak** and **wbak** in a mixed environment. First, backing up an SR10 node with **wbak** from an SR9.7 node works correctly so long as the names of the objects on the SR10 node conform to the SR9.7 restrictions with respect to pathname and component name length.

If you attempt to restore an SR10 format tape with the standard SR9.7 **rbak**, serious errors will occur. To restore SR10 objects to SR10 volumes from SR9.7 nodes, use the **rbak** command installed in the **/sr9.7_executables** directory. This version of **rbak** prints an error message if it encounters a tape format that it does not recognize. You'll also need to be sure that this command resides on any SR9.7 node that needs to use **rbak** to restore objects to SR10 nodes.

To create a tape on an SR10 node that you can read on an SR9.7 node, use the **-presr10** flag with **wbak**. This tape will have no ACLs by default. To restore the tape to an SR9.7 node, use a pre-SR10 version of **rbak**. If you make a tape on an SR10 node without the **presr10** option, you will not be able to restore that tape successfully to a pre-SR10 system.

Because there are no canned ACLs in SR10, ACL entries for **%.backup.%** are no longer given to directories (and files) by default at installation. This allows you flexibility in the way you perform backups on your network. To operate as you did before SR10, add ACL entries for **%.backup.%** to file system objects that you wish to back up.

If you wish to perform backups in an alternate way, say, running as root, you can do that without adding ACL entries. You do not need to perform backups as root. See the *Managing System Software* books for specific suggestions about performing backups as root, as well as information about the effects of **lprotect** and the LOCAL_ACCESS_ONLY attribute on running backups as root.

Before SR10, the operating system used a mechanism we called colon-mapping to distinguish uppercase letters (and a few special characters) in the case-insensitive Aegis environment. A file you created in a Domain/IX environment named ***Readme** would appear, in an Aegis shell, as **:readme**.

The SR10 versions of the **rbak** and **wbak** commands automatically map colon-character constructs (pre-SR10 capital letters) into the appropriate character. Therefore, if you wish to preserve literal colons in existing pathnames, you should use the **cvtname** tool. In an SR10 file system, the colon has no meaning as an escape character.

Mixed Networks and uucp

Because pre-SR10 Domain/IX systems use a different version of **uucp** than the HoneyDanBer version shipped with SR10, pre-SR10 **uucp** cannot share spooling queues with SR10 **uucp**. In order for you to upgrade your **uucp** site node to SR10 and still interoperate with pre-SR10 nodes, we provide an SR9.7 version of HoneyDanBer **uucp** to run on SR9.7 nodes only. Installing this on SR9.7 nodes allows SR9.7 and SR10 nodes to share a common spooling queue. This version of HoneyDanBer **uucp** is in the directory **/sr9.7_compatibility/sr9.7_executables**.

Usually only one node in an Apollo network is configured as the **uucp** site node. Other nodes on the network have their **/usr/spool/uucp** and **/usr/lib/uucp** directories linked to the **uucp** site node. In a network of mixed pre-SR10 and SR10 machines, only machines running HoneyDanBer **uucp** will interoperate. Therefore, you must have the SR9.7 version of HoneyDanBer **uucp** running on any SR9.7 nodes that you want to be able to communicate with the SR10 HoneyDanBer site node.

If the site node is upgraded to SR10 HoneyDanBer **uucp**, the SR9.7 HoneyDanBer **/$(SYSTYPE)/usr/bin uucp** files should be installed on any SR9.7 user nodes you want to run **uucp**. If the site node is running SR9.7, the SR9.7 HoneyDanBer **/$(SYSTYPE)/usr/lib/uucp** and **/$(SYSTYPE)/usr/spool/uucp** files should be installed. Note that the configuration files must be converted from the old names and formats to the new ones. This means that, once you've upgraded to HoneyDanBer **uucp**, pre-SR9.7 nodes on your network will not be able to use the old version of **uucp** to communicate. Refer to the *Managing System Software* books for further details on converting configuration files. Refer to the online file **/sr9.7_compatibility/sr9.7_executables/uucp_README** for installation instructions.

Through modems, however, the SR10 version of **uucp** will communicate with pre-SR10 Apollo **uucp**, as well as other vendors' versions of **uucp** and HoneyDanBer **uucp**.

Mail Address Registry Field

The mail gateway field (which uses the SR10 registry for storing information) has no effect when you run with a read-only SR10 registry.

Print Services

The new Aegis print architecture works transparently in a mixed environment that contains SR10 and SR9.7 print servers. Two information request options, **–pre10** and **–check**, have been added to the **prf** command to provide additional control in mixed environments.

By default, the SR10 **prf** command first submits a print job using the SR9.7 format (it creates a file in **/sys/print/queue** and copies the data file to **/sys/print/spooler**). The SR10 **prf** command then queries the print managers on the network to find the specified printer name. If the name is found to be an active SR10 printer, the job is added to the print manager's queue. If the specified printer does not exist, **prf** returns with no error message, assuming the job has been queued to an SR9 print server.

The **–pre10** option eliminates the query to the print manager. Use this option only when you are sure there are no SR10 print managers on the network.

The **–check** option is used if you want to be sure you are queuing a job to an SR10 printer. Adding the **–check** option to your command line or **startup.prf** file will cause **prf** to report an error and not queue the job if the specified printer name is not an active SR10 printer. This option will become a default in a future release.

Ordinarily, in a mixed environment of SR10 and pre-SR10 systems, the **/sys/print** directory must be located on an SR9.7 node; otherwise, users on SR9.7 nodes cannot queue jobs. However, the **/sr9.7_compatibility/sr9.7_executables** directory contains versions of **/com/prf** and **/lib/prflib** that overcome this restriction. If you install these items on the SR9.7 nodes in place of the original **prf** and **prflib**, you can then put the **/sys/print** directory on an SR10 node and queue jobs successfully from the SR9.7 nodes. In all cases, the print manager and print server can run on SR10 or SR9.7 nodes.

If you find it necessary or desirable to put the spool directory on an SR10 volume, then you must copy **/lib/prflib** and **/com/prf** from the **/sr9.7_compatibility/sr9.7_executables** directory to the SR9.7 nodes on your network and reload the libraries (by rebooting or exiting the Display Manager).

The new features of the SR10 architecture are, of course, not supported on an SR9.x node.

File System Incompatibilities

There are some incompatibilities between the file systems of SR10 and pre-SR10 operating systems.

Object File Formats

The format for executable files changed at SR10. Before SR10, the system used the **obj** format (Apollo object file format). At SR10 and beyond, the system uses an extended version of the AT&T Common Object File Format (COFF) standard. (For additional information on the COFF standard, see Chapter 4.)

Because of the difference in object file formats between SR9.x and SR10, be careful about which object modules you try to run on which nodes. Any SR9.2 object modules that run on SR9.5 (and were not compiled for **sys3** or **bsd4.1**) will run on SR10, assuming they meet other compatibility restrictions like case sensitivity and longer names. However, pre-SR10 machines will not be able to run SR10 (COFF) object files at all.

You should consider these restrictions when setting up links between nodes, as well.

Be aware of type managers compiled as COFF objects, since they may get passed between nodes and find their way to pre-SR10 nodes. Type managers compiled as obj objects do run on SR10 nodes. However, in mixed networks, type managers that reside on an SR10 node may not always be loaded dynamically across the network to a pre-SR10 node, since SR10 type managers are compiled as COFF modules, and COFF objects do not run on pre-SR10 nodes.

This problem can occur when you use products like Domain/Access Release 3.0, since type managers included with this product are compiled as COFF objects. See the Release Document for Domain/Access Release 3.0 for complete details.

Layered products that require you to compile programs are also subject to the limitations on COFF and obj objects in mixed networks. For example, with Domain/LU6.2 Release 1.1, a Transaction Program (TP) that you compile by using an SR10 compiler won't run on a pre-SR10 node because the program is a COFF object. For further details about using Domain/LU6.2 Release 1.1 in a mixed network, see the Release Document for this layered product.

Command Search Rules

From a pre-SR10 node, if you have an SR10 node in your command search rules (CSR) or PATH, you'll have problems because you can't run COFF object modules on pre-SR10 systems, and all commands at SR10 are compiled with COFF.

Be aware of the effects of variant links in a mixed network, especially the **$(SYSTYPE)** link. On an SR10 node running BSD, for example, the *$(SYSTYPE)*/**usr** directory will resolve to **/bsd4.3/usr**. If you want to maintain, say, **bsd4.2** directories on a directory, add the pathname for the **bsd4.2** trees to the PATH or CSR explicitly, or specify the pathname in full every time you try to gain access to the trees. If you run SR9.7 programs on SR10 nodes, you must be aware that the SYSTYPE variable may not be what the program expects.

Differences in Name Resolution

The major differences in name resolution that were just detailed have a significant impact in a mixed environment. The potential problems in ignoring case when you're operating in a case-sensitive environment are obvious, but what may be more subtle is how changes in the tilde, backslash, and other special characters affect users in mixed networks.

For example, if you type **cat ~file** on a pre-SR10 machine, you'll see the contents of the file in your naming directory named **file**. If you type the same thing on an SR10 node, you'll get "No such file or directory," assuming you don't have a file literally named **file**. The correct syntax for specifying a file in the naming directory is **~/file**. You'll have to make users aware of the probable ill effects of this and other constructions in shell scripts, search rules, and programs, as well.

Remote Process Creation

For you to create a remote process (**crp**) on an SR9.7 node from an SR10 node and have the improved features of the SR10 **crp** command, the type **spmio** must exist on the SR9.7 node. The type is installed in **/sys/mgrs** on an SR10 node by the SR10 installation procedure, and must be installed manually on the SR9.7 node.

To install this type, use the **inty** shell command, as shown below. (If you don't install the type, **crp** continues to work as it did at SR9.7.)

inty spmio *source_volume* **-n** *node_spec*

The *source_volume* argument tells from which SR10 node to copy the type name and manager. The *node_spec* argument specifies the target, or SR9.7, node.

When you use **crp** to log in to a pre-SR9.7 node from an SR10 node, you'll have difficulties using the **-me** option. This is because the **crp -me** command sequence sets your naming and working directories and, if either of those directories is an SR10 directory, the pre-SR9.7 node cannot resolve it. Invoke **crp** without **-me** and log in to avoid this problem.

Using emt in Mixed Networks

SR10 includes a version of the **emt** utility for users who wish to **crp** from an SR10 node onto an SR9.x node and use **emt** on the SR9.x node. This version of the **emt** utility resides in the following directory: **/sr9.7_compatibility/sr9.7_executables/com**.

Layered Product Incompatibilities

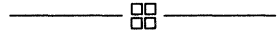
The change to COFF format introduces incompatibilities with the Network License Server.

Network License Server

If your NLS application was built as an obj format object module (that is, with pre-SR10 tools), it operates correctly on both SR10 and pre-SR10 nodes in a mixed network. However, if your application was built as a COFF object, you must run the **obj2coff** tool over the library **/sys/nls/nls_lib** for the application to run on SR10 nodes.

Domain/IX: SR10 Incompatibilities

Any incompatibilities between Domain/IX (that is, pre-SR10) and SR10 UNIX environments, other than those we've already discussed, are limited to differences between System V Release 2 and System V Release 3, and between 4.2BSD and 4.3BSD. These incompatibilities are reflected in the standard documentation for those UNIX systems.



Chapter 4

Implications for Programmers

This chapter addresses the implications for programmers of changes to the SR10 operating system software.

General

In general, changes to the programming environment at SR10 fall into the same areas as other parts of the operating system. These areas include case sensitivity, different naming rules, and changes to protection.

We've made a number of changes specific to the programming environment, including new compilers, new default object file format (COFF), absolute code, single program-per-process model, and ANSI C function prototypes.

Compatibility

The SR10 BSD and SysV environments are based on 4.3BSD and System V Release 3 UNIX systems, respectively. The commands, bindable libraries, and include files originated with the UNIX 4.3BSD and System V Release 3 versions.

System V Release 2 and System V Release 3 are generally forward compatible, but 4.2BSD and 4.3BSD are not, completely. If your programs

rely on Domain/IX 4.2BSD or System V Release 2 features (that is, system types of **bsd4.2** or **sys5**), you should copy the appropriate 4.2BSD and System V Release 2 **/usr/include** directories, under the **/bsd4.2** and **/sys5** directories, from an SR9.7 node.

The **bsd4.1** and **sys3** system types are obsolete.

SR10 Library Model

The scheme for loading program libraries has changed at SR10. By using a per-node configuration file, **/etc/sys.conf**, you can specify which libraries are to be treated as "global, " that is, loaded into global space at node boot time, and which libraries are to be treated as "shared," that is, loaded into the private address space of a program at run time. This distinction is especially useful for systems with small virtual address spaces (16 MB and 64 MB) that do not have enough room in global A space to load all libraries.

Shared libraries (those not marked "global") may be loaded either statically or dynamically. That is, they may be loaded when a program containing a reference to the library gets loaded, or loaded only when the executing program actually executes a call on the library. A dynamic library is marked "dynamic" in the **sys.conf** file. Note that a library marked "global" cannot also be "dynamic."

All libraries specified in **/etc/sys.conf**, whether they are global or shared, have their entry points inserted into the Known Global Table (KGT) of global A at boot time. Any new versions of these libraries copied into **/lib** are visible until the node exits back to the boot shell or shuts down.

Libraries can also be designated "optional," which means that no error will be reported if the library is not found. Additional flags in **/etc/sys.conf** allow you to specify the smaller-virtual-address-space systems for which certain libraries are not to be global.

In the absence of the **sys.conf** file, a default set of libraries is always loaded. Some libraries not marked "global" may get loaded globally anyway, if they are referenced by libraries that are loaded globally.

The Bind Utility

We've made the following changes to the bind utility for SR10:

- The binder now handles only COFF objects; that is, objects produced by SR10 compilers, SR10 linkers (**bind** or **ld**), or SR10 archivers (**lbr** or **ar**). The binder does not handle any object files compiled prior to SR10.

- The **bind** utility now supports both a **-runtype** option to control the UNIX run-time system call semantics and a **-systype** option to control pathname semantics. The SR10 compilers also support both the **-runtype** and **-systype** options.

- At SR10, you must use the **-allocbss** bind option whenever you compile a C program with either **/bin/cc** or with **/com/cc** and the **-bss** option. The **-allocbss** option allocates space for globals, and is required, even if the C program consists of only one source file. When you bind with **-allocbss**, you cannot run the output file through the binder again.

If you compile by using **/bin/cc –W0 –nbss** or **/com/cc** (without the **–bss** option), you should use **–mergebss** to merge the resulting multiple global data sections. Conversely, if you compile with **/bin/cc** or **/com/cc –bss**, you *must* use **–allocbss** in the final bind operation to create a single data section and to put all the uninitialized global variables (which are not assigned a section during compilation) into the section. Unlike **–mergebss**, the **–allocbss** option does not merge any existing uninitialized global data sections into the **.bss** section.

The lbr Utility

We've made the following changes to the librarian utility, **lbr**, for SR10:

- The **lbr** utility handles only objects generated by SR10 compilers, SR10 linkers (**bind** or **ld**) or SR10 archivers (**lbr** or **ar**). The **lbr** utility now generates library files in the

form of UNIX archive (**ar**) files. For compatibility, we provide a version of **lbr** that handles objects created between SR9.5 and SR9.7. This command resides in **/sr9.7_compatibility/compat_with_sr9.7/com**. Note that the **lbr_sr9.2** utility is now obsolete.

- The format of **-list** output is now the same as **ar -tv** format.

The tb -args Command

Because COFF objects do not have the information necessary to support the **-args** to **/com/tb**, the option has been removed.

The /bin/ld Command

Because **/bin/ld** no longer invokes **/com/bind**, you must be aware of different search rules when you construct a **/bin/ld** command line. At SR10, **/bin/ld** searches libraries in the order they appear on the command line.

Libraries

The library **/lib/unixlib** is obsolete at SR10. It has been replaced by **/lib/clib** and **/lib/libc**.

We've provided libraries with the SR10 release that you can use to run the SR10 compilers on SR9.7 nodes. They reside in **/sr9.7_compatibility/sr9.7_executables/lib**. You must copy the following libraries from an SR10 node to the SR9.7 node that you want to run the SR10 compilers on:

/lib/kg_.lib
/lib/name_.lib
/lib/loader_.lib

The **/sr9.7_compatibility/sr9.7_executables** directory contains an SR9.7 version of **/lib/streams** to correct an error you may encounter when running in a mixed network. If you run a program on an SR9.7 node that uses the **ios_$close** call to close a temporary file that resides on an SR10 node, the call returns the error status "f0001, object not found." The operation has correctly occurred on the SR10 node, but the return of that status code may cause the program to fail. Install the **/lib/streams** from

/sr9.7_compatibility/sr9.7_executables/lib/streams on the SR9.7 node to correct this behavior.

Please note that this version of the streams library supports only Version 3.1 of TCP/IP. If you are running Version 3.0 of TCP/IP, you should not install this version of **/lib/streams** on your SR9.7 node.

The /lib/prflib Library

Any applications that bind a pre-SR10 version of the library **/lib/prflib** must rebind with the SR10 **prflib**. Use the **-inlib** option to **bind** to accomplish this.

Insert Files

There have been changes to insert files at SR10.

Location

At SR10, all system insert files reside in the directory **/sys/ins**. The C function prototypes reside in the directory **/usr/include/apollo**.

C Insert Files

At SR10, we provide two sets of include files for the C language, the current **/sys/ins/*.ins.c** files and the new **/usr/include/apollo/*.h** files (for example, **ios.h, gpr.h**).

The ***.h** include files take advantage of the new prototyping and reference variable features of Domain C. Together, these features simplify cross-language communication and make the **std_$call** convention obsolete. We recommend that you use the new include files in all new C programs.

The include file **/usr/include/apollo_$std.h** is currently supported for use with the **/com/cc** C compiler. Future releases of the C compiler may not include this file.

Implications of Obsolete System Types

Before SR10, if you did not specify a SYSTYPE at compile time (or inside the C program itself), or if no SYSTYPE environment variable existed, the C compiler stamped the object module with a SYSTYPE of **sys3**. This means, for example, that many applications compiled with **/com/cc** before SR10 are stamped **sys3**.

The SYSTYPEs **bsd4.1** and **sys3** are obsolete at SR10, and the SR10 loader does not load programs marked with these SYSTYPEs. In the case of C programs given a SYSTYPE of **sys3** by **/com/cc**, you can either recompile, checking to make sure the programs are upward compatible, or you can rebind with a new SYSTYPE.

The SYSTYPE "any" continues to be valid. However, applications marked with a SYSTYPE of "any" must be compiled with that SYSTYPE explicitly. This was true before SR10 and is true now.

Transition from Obsolete System Types

As stated in the previous section, programs with SYSTYPEs of either **sys3** or **bsd4.1** will not run at SR10. If your site does not have access to the source code for these programs, you may try rebinding the programs with a new SYSTYPE (either **sys5** or **bsd4.2**, as appropriate). The programs *may* run without other changes in the new environment; we can offer no guarantees. You cannot rebind to SYSTYPEs of **bsd4.3** or **sys5.3** because of changes made at SR10, most notably changes made to the **stat** structures.

The ios_$ Interface

As a side effect of work on the UNIX I/O system calls, **ios_$** calls have become interruptible. This is different behavior from SR9.7. You are most likely to encounter this difference with the **ios_$get** and **ios_$put** calls.

The pad_$ Interface

We've added a new call, **pad_$isa_dm_pad**, which tells you whether a stream is a Display Manager pad. See the online help pages for Domain/OS system calls for details.

The mkdir(2) System Call

We've made changes to the **mkdir** system call to alter the way a directory inherits protections. When you use **mkdir** in a program to create a tree now, the new directory inherits its initial file and directory ACLs from its parent directory; that is, **mkdir** operates the way the **crd** command would. This change in the **mkdir** system call enables directories to propagate protection semantics to their subdirectories.

Options to the **chacl** command allow you to force a particular directory to have UNIX inheritance characteristics. See the online manual page for **chacl** for details.

The open(2) System Call

At SR9.7, the O_CREAT flag to an **open** call implied write access to the file opened. This behavior conflicted with the SVID, and has been changed at SR10. You should always include one of O_RDONLY, O_WRONLY, or O_RDWR when you perform an **open** in a program.

The fst Command

The **fst** command, which prints fault status information, works only if the INPROCESS environment variable is set and all commands are run in-process. For the normal SR10 environment (that is, not in-process), use **tb** to get the information that **fst** used to produce.

Case Sensitivity

At SR10, the kernel and libraries are fully case sensitive. Programs that depend on (or perpetuate) old, case-insensitive behavior do not work at SR10, unless you use the temporary transition aids we have supplied.

In this section, we supply a few simple rules that, when applied to your programs, ensure that they behave properly in the case-sensitive world of SR10. (Individual programming languages, of course, have their own naming rules; we have not changed any of these.)

Pathnames

Don't manipulate case in pathnames. Assume that pathnames, whether supplied by users, programs, or system calls are supplied case correct. A request to open "/myDIR/MYfile," whether it comes from a user or another program, must not be changed into a request to open "/MYDIR/MYFILE" or "/mydir/myfile." The corollary to this is that programs that emit pathnames (to users or to other programs) must never emit a pathname that is not case correct. At SR10, attempts to open "/SYS/NODE_DATA/FOO" are guaranteed to fail, unless the user has set DOWNCASE to **true**.

Programs that use the **%ua** or **%la** case conversion options in the control string of **vfmt** calls that print out pathnames should be changed to use the **%a** option instead.

Be sure to express all constant pathnames embedded in source in the correct case. Typically, this means lowercase (for existing non-Domain/IX software), although it is not uncommon for UNIX programmers to use mixed-case leaf names (for example, **/usr/lib/getNAME, /usr/lib/font/ftR**) and even to create names that are identical in all ways except case. Programs containing embedded pathnames that are not case sensitive must be modified to work at SR10. For example, a program that contains the definition

```
CONST my_file = '/SYS/MY_FILE';
```

doesn't work at SR10 because the **/sys** directory is named **/sys**, not **/SYS**. The definition

```
CONST my_file = '/sys/my_file';
```

does work at SR10 (assuming, of course, that the file name was **my_file**, and not **MY_FILE**.

Calls that Return Names

All calls that return names should return them case correct. The following case-correct calls are obsolete:

name_$get_wdir_cc();	Read the working directory of the calling process in the correct case.
name_$get_ndir_cc();	Read the naming directory of the calling process in the correct case.
name_$read_dir_cc();	Read directory entries in the correct case.
name_$get_path_cc();	Find the FULL pathname of the specified object in the correct case.

These have been replaced by calls that support both long names and case-sensitive behavior.

All **name_$create** calls return "name_$already_exists" if the name exists. This semantic is unchanged at SR10.

The **name_$delete** calls do not reliably return "name_$not_found." A call may return "status_$ok," even if the name did not previously exist. Therefore, you cannot assume that the object did exist if you delete it and get "status_$ok." You will get "name_$not_found" reliably for local objects. See "New Interfaces" under the section on long path and component names later in this section.

Symbol Names

At SR10, the C compiler emits case-correct symbol names. The Pascal and FORTRAN compilers emit lowercase names. If you need to share data between C and Pascal or between C and FORTRAN, be certain that global variables are declared in all lowercase letters in the C routines.

Transition Aids

The following subsections summarize transition aids that we supply to help users adapt to a case-sensitive environment at SR10.

Downcase Mode

Downcase mode affects the way the naming server interprets pathnames that are presented to it. Users control downcase mode by means of a per-process environment variable called DOWNCASE. If DOWNCASE is **true**, then a process runs in downcase mode.

We do not recommend relying on the DOWNCASE environment variable or any other means of running in downcase mode after SR10. It is a transition aid, and support will be removed in a future release.

When a process is running in downcase mode, the naming server's behavior changes in two ways:

- It forces to lowercase all pathnames presented to it as input before it tries to resolve them.
- It treats the set of characters

  ```
  ~`\.()
  ```

  as syntax, rather than as constituent characters of a name. (This is compatible with the pre-SR10 default setting of the NAMECHARS environment variable. Any attempt by the user to set NAMECHARS is ignored at SR10.)

Files with mixed-case names are inaccessible to programs running in downcase mode, although the DM escape mechanism of delimiting mixed-case names with single quotes continues to work.

When a program is not in downcase mode (the default at SR10), the naming server treats all characters in pathnames as constituents, with the exception of the following:

`

The tic (backquote) in the name **`node_data** remains unchanged from previous releases.

~

The tilde is now a reserved file name which the naming server always expands into the name of the current naming directory. We no longer support the notation

```
~foo
```

as a shorthand for "naming_dir/foo". Instead, the notation is now

```
~/foo
```

$(*name*) We have retained the variable link mechanism. When the naming server sees the dollar-sign open-parenthesis string, it treats everything between the open parenthesis and the next closing parenthesis as a variable name, expanding it to the current value of that variable name.

The backslash and dot characters have different syntactic meanings in the default SR10 environment, unless a process is running in downcase mode. Backslash no longer represents the parent directory. Dot, dot-dot, and tilde all work as before when followed by a slash. Otherwise, they are interpreted as literal characters in the name.

The following table summarizes changed interpretations of dot, dot-dot, backslash, and tilde.

Name	Meaning	
	Pre-SR10	*at SR10*
\foo	name "foo" in the parent of this directory	name "\foo" in this directory
../foo	name "foo" in the parent of this directory	no change
.foo	name "foo" in this directory	name ".foo" in this directory
./foo	name "foo" in this directory	no change
~foo	name "foo" in naming directory	name "~foo" in this directory
~/foo	name "foo" in naming directory	no change

At SR10, the notation

```
\foo
```

refers to the name **\foo** in the current directory, rather than to the name **foo** in the parent of the current directory. To specify the name **foo** in the parent, you need to say

```
../foo
```

You can also no longer use a file's pathname as the root of a pathname with backslashes. For example, in an Aegis shell script, the following construction is not valid, while the backslash equivalent was valid at SR9.7:

```
^0/../program2
```

where **program2** is another file in the same directory as the script. In addition to changes in the backslash notation, the notation

```
.foo
```

refers to the name **.foo**, rather than to the name **foo**, in the current directory. To refer to a **foo** in the current directory, you have to say

```
foo
```

or

```
./foo
```

These changes in syntactic rules are in effect for all programs not running in downcase mode. Programs running in downcase mode can use the current interpretations of backslash and dot.

The case Command

The Display Manager (DM) command **case** allows you to change the case of letters in a selected range of text. You can use this command to change uppercase letters in source code or shell scripts to lowercase. See the online manual pages for additional information.

Summary of Incompatibilities

The SR10 move to case sensitivity introduces the following incompatibilities for pre-SR10 programs and shell scripts:

- Programs that alter pathnames supplied by users, system calls, or other programs may not be able to access the objects to which those pathnames refer.
- Programs that do not use case-correct name-returning calls may not be able to access the objects to which pathnames returned by those calls refer.
- Programs and shell scripts that contain case-incorrect embedded constant pathnames cannot access the objects to which those pathnames refer.
- Shell scripts that use any of the following notations behave differently:

  ~name
  .name
  \name

- Programs that store colon-mapped file names in files cannot find those names in an SR10 file system.

In order to work correctly in the new environment,

- Programs must not arbitrarily alter the case of pathnames.
- Programs must not contain case-incorrect embedded pathnames.
- Programs must not depend on the pre-SR10 interpretations of dot, backslash, and tilde when these characters appear as the first character of a pathname.

We provide downcase mode as a compatibility mechanism at SR10. However, programs that run in downcase mode cannot resolve mixed-case pathnames and may exhibit other behavioral anomalies. We plan to support downcase mode for no more than one major release of system software.

Invoking Programs

At SR10, all shells invoke new programs as new processes so as to support **absolute**, or non-relocatable, code (code compiled to load at a fixed address in memory). As a compatibility mechanism, an environment variable (INPROCESS) can cause the system to invoke programs in the same process (in the Aegis **/com/sh**). When we refer to **in-process** invocation, we mean the pre-SR10 mechanism of invoking a new program at a new program level within the same process. (This mechanism requires program code to be position independent.)

When we refer to **extraprocess** invocation, we mean the UNIX mechanism of invoking a new program as a new process.

In addition, we have maintained support for the mark-release mechanisms that clean up resource usage at each in-process program level (unmapping files, freeing storage) However, the mark-release support will be removed at the next major software release.

In-process invocation will continue to be available, but will not be supported by the mark-release mechanism. Therefore, only carefully written programs will find in-process invocation useful after SR10. We strongly urge programmers to move to the single-program-per-process model.

Clean-Up Handlers

At SR10, a program's clean-up handlers must be released at each program level. The **pfm_$cleanup** call does not automatically release the clean-up handler at the end of normal execution. If your program issues a **pfm_$cleanup** call, it must also issue a **pfm_$rls_cleanup** call at the end of normal execution.

Number of Streams (File Descriptors) Open

Before SR10, the standard environment for Aegis programs included four file descriptors that were open when a program started executing. These descriptors, 0, 1, 2, and 3, were named standard input (stdin), standard output (stdout), error in (errin), and error out (errout), respectively. UNIX programs, on the other hand, expect three open file descriptors, 0, 1, and 2, named standard input, standard output, and standard error, respectively.

Prior to SR10, both Aegis and Domain/IX programs were invoked with four standard file descriptors, which occasionally caused Domain/IX programs to fail.

At SR10, both Aegis and UNIX programs are invoked with three file descriptors, or streams, open; that is, COFF objects are invoked with three streams open. Obj objects (pre-SR10 format) are still invoked with four streams. The rules are summarized as follows:

- When a COFF object invokes another COFF object, or an obj object invokes another obj object, the **pgm_$invoke** or **exec** call passes through all the streams specified by the caller (3 for COFF, 4 for obj).

- If a COFF object is invoked by an obj program and passed four streams, the invoking process will close stream 2 (errin) and move stream 3 (errout) to stream 2.

- If an obj object is invoked by a COFF program and passed only three streams, the **pgm_$invoke** or **exec** call copies stream 2 (standard error) onto stream 3, assuming that stream 3 is not already open.

Some commands in the **/com** directory are in obj format at SR10, so that shell scripts will continue to work correctly. The Aegis shell, **/com/sh**, is an obj format object and therefore receives four streams. It also passes four streams in all cases but one: if errin is redirected for the command that **/com/sh** is invoking, and if the command being invoked is a COFF object, an error is returned.

At SR10, the calls **ios_$errin** and **stream_$errin** are undefined. Your programs should move to the three-streams model. This means that references to errin in your programs should be purged. Below are two ways to remove these references:

- Many programs have used errin in the past to ensure that a program is reading from the keyboard. A more reliable way to do this at SR10 is to have the program open **/dev/tty**.

- For other uses of errin, you can change references to errin in the program to read from **ios_$stderr**.

Relocatable and Absolute Code

By default, the SR10 compilers generate absolute code, which the loader loads at 0x8000. Most Apollo commands use the absolute code. However, the compilers retain the ability to generate position-independent code. You must produce position-independent code for all routines placed in a global or dynamically loadable library.

Program Invocation Semantics

The default semantic of **pgm_$invoke [pgm_$wait]** changes from in-process to extraprocess invocation unless the INPROCESS environment variable is set. The table below shows the relationship of the *mode* argument of **pgm_$invoke** to the value of INPROCESS.

Mode	INPROCESS	What Happens
"" (empty)	set	invokes extraprocess, continues
"" (empty)	unset	invokes extraprocess, continues
pgm_$wait	set	invokes in-process, waits
pgm_$wait	unset	invokes extraprocess, waits

UNIX shells use **fork** or **vfork**. The following table shows the effect of INPROCESS on pre-SR10 (specifically SR9.5) and SR10 mechanisms for program invocation.

Shell	at SR9.5	at SR10
/bin/sh	pgm_$invoke[pgm_$wait] (implicitly in-process)	fork()/exec()
/bin/csh (inprocess set)	pgm_$invoke[pgm_$wait] (implicitly in-process)	vfork()/exec()
/bin/csh (inprocess unset)	vfork()/exec()	vfork()/exec()
/com/sh (inprocess set)	pgm_$invoke[pgm_$wait] (implicitly in-process)	pgm_$invoke[pgm_$wait] (implicitly in-process)
/com/sh (inprocess unset)	pgm_$invoke[pgm_$wait] (implicitly in-process)	pgm_$invoke[pgm_$wait] (implicitly extra-process)

Inheritance Rules

Most pre-SR10, parent-to-child inheritance rules still apply. A child process automatically inherits its parent's real and effective SID, environment list, and list of pending and blocked signals.

If a program is invoked in-process, a child process inherits streams opened in the parent program even if the parent does not include any code that explicitly passes the open streams to the child. Programs that rely on this behavior will not work at SR10 unless the INPROCESS variable is set to **true**. The INPROCESS transition aid with mark-release will be supported for one major release of system software.

Programs Affected

We expect the following types of programs to require changes in order to work properly in a single-program-per-process world:

- Programs that expect to be able to share address space, in any direction, across program invocation levels
- Programs that expect to share any open streams with their children and do not make explicit arrangements to do so

Transition Aids

If the environment variable INPROCESS is set, programs are invoked in-process. This transition aid is supported with the mark-release cleanup for SR10 only.

Object Module Format

All SR10 Apollo compilers generate a new object format which is an extended version of the AT&T COFF (Common Object File Format) standard. The loader retains knowledge of how to load old (Apollo pre-COFF) objects; however, it is not possible to bind old and new modules together. In addition, library files use UNIX **ar** format.

Be aware that COFF object files only run on SR10 nodes. For example, if you have COFF files in your **~/com** directory and you are logged in on an SR9.7 node, you will not be able to run them successfully.

The **crtyobj** command, which creates an object module to be bound with a type manager, only creates COFF modules. If you need to create pre-SR10 format object modules for type managers, use an SR9.7 version of the **crtyobj** command.

Many of the programming tools released at SR10, including **dbx**, **ld**, and **ar**, do not understand the "obj" format. Although you will be able to execute obj format files on SR10 nodes, you do not have the same level of access to these files that you have to COFF files. It is not possible to bind old (obj) files to new (COFF) files.

All programs that read object modules must be endowed with the ability to read COFF objects as well as (or instead of) old Apollo objects.

Transition Aids

We have provided the conversion programs **obj2coff**, which converts old Apollo object files to COFF, and **lbr2ar**, which converts old Apollo libraries to **ar** format. Both of these programs reside in the **/usr/apollo/bin** directory. These programs should postpone, if not eliminate, the need to recompile.

Converting obj Modules to COFF

The **obj2coff** tool converts a binary file from obj format to COFF. See the online manual pages for further details.

Converting Library Files to ar Format

The **lbr2ar** command converts pre-SR10 **lbr** libraries that contain obj object modules into **ar**-style archives that contain COFF object files. The **lbr2ar** command invokes **obj2coff** and expects to find it in the **/usr/apollo/bin** directory. The **lbr2ar** supports one option, which allows the user to specify an alternate directory where **obj2coff** resides.

The command syntax is as follows:

lbr2ar [-Y/xx/yy] *lbr_file ar_file*

In the example above, instead of calling **/usr/apollo/bin/obj2coff**, **lbr2ar** would call **/xx/yy/obj2coff**. See the online manual pages for further details.

Mixed Networks and COFF Modules

If you create a COFF module on an SR9.7 node and place it on an SR9.7 or SR10 volume, the COFF file will appear to be an ASCII readable file, and the Display Manager (SR9.7 or SR10), upon request, attempts to open a pad on it. This is also true for a COFF file that you deposit on an SR9.7 volume from an SR10 node.

We have also included in SR10 a command that can modify a COFF module so that the DM will not open a pad on it. The command is **make_bin**, and it is found in the directory called **/sr9.7_compatibility/compat_with_sr9.7/com**.

Long Path and Component Names

At SR10, we increased the maximum length of a component (leaf) name from the old limit of 32 characters to a new limit of 255 characters, and increased the maximum length of a pathname from the old limit of 256 characters to a new limit of 1023 characters. Prior to SR10, variable names were unique through the first 32 characters. At SR10, variable names are unique through 4096 characters.

We made these changes along with implementing a new, more robust, higher-capacity directory structure. Any program that uses Aegis system calls that expect or return names may be affected.

Summary of Changes

To support long names, we created the data types **name_$long_name_t** and **name_$long_pname_t** and a series of calls with the suffix **_lc** (long case-correct) that use these data types in place of the pre-SR10 **name_$name_t** and **name_$pname_t**.

New Data Types

The following new data types are available at SR10 in support of long names:

Type	Size	Defined in
name_$long_name_t	256 bytes	base.ins.*
name_$long_pname_t	1024 bytes	base.ins.*

Since every name is null terminated, actual maximum lengths are 255 characters for a leaf name and 1023 characters for a pathname.

New Interfaces

The following new interfaces were released at SR10:

name_$get_wdir_lc
name_$get_ndir_lc
name_$get_path_lc
name_$read_diru_lc
name_$read_link_lc
name_$extract_data_lc

The following unreleased calls are not available to all customers; they are also new at SR10:

name_$read_dir_lc
name_$gpath_lc
name_$get_entryu_lc
name_$strip_leaf_lc
name_$read_linku_lc
name_$find_uid_lc
name_$resolve_afayc_lc
name_$inq_dir_lc

These interfaces are functionally identical to pre-SR10 interfaces with similar names (but without the **_lc** suffix). The differences are that the new interfaces do the following:

- Return **name_$long_name_t** instead of **name_$name_t**, and/or **name_$long_pname_t** instead of **name_$pname_t**.
- Expect the caller to specify an output buffer length (the maximum size of the name the caller expects to get back) and return an error if the size of the returned name exceeds that length.

Transition Aids

We have provided a tool, **/etc/show_lc**, that examines an object module and lists any calls the module makes that may be affected by the move to long names.

Programmers must audit their own code for procedures in which a data type has been defined as a **name_$name_t** or a **name_$pname_t**.

Summary of Incompatibilities

At SR10, it is possible to create longer path and component names than with previous releases. Programs using interfaces from previous releases return an error when attempting to read the longer names.

We have released new versions of name-returning interfaces, along with a tool that examines object modules for references to old name-returning interfaces.

In order to support long names, customers must use the tool to examine any object module (of their own creation) that they suspect returns a name, change their source code to use the new interfaces, and recompile.

Although such modifications are not a requirement, unmodified programs return an error when they attempt to read a long name.

Long Symbol Names in Libraries

The obj format supports symbol names up to 32 characters long, but COFF supports longer names. When the obj C compiler encounters a name longer than 32 characters, it gives a warning message and truncates the name. The obj Pascal compiler truncates without providing a warning. This can become an issue when programs and the libraries they access are in different object file formats. For example, a COFF program would try to call a procedure with its complete long name, but the procedure's name in an obj format library would be truncated to 32 characters, and the call would fail.

At SR10, we've solved this incompatibility by doing two things. First, any libraries that may be affected by this problem are shipped as COFF objects. This ensures that COFF programs run correctly. In addition, we've provide 32-character stub names in these libraries (for example, **gm_$viewport_set_background_valu** is used to substitute for **gm_$viewport_set_background_value**) so that obj programs are able to access the symbols too.

ACLs

At SR10, user and system interfaces to the Access Control List (ACL) manager interface have changed in ways that introduce some incompatibilities for both programs and users.

We made these changes to allow full support for the UNIX protection model and all its semantics. Accordingly, we have limited the number of rights we support to the UNIX **rwx** (read, write, execute) set, along with two extensions: **p** (the right to change protection), and **k** (keep; do not delete or change the name of the object).

Summary of Changes

At SR10, we require every object in the file system to carry rights specifications for *owner*, *group*, and *world* rights (the UNIX model), as well as for *organization* (from Aegis). These required rights are maintained as part of every file. Additional protection specifications, if needed, are stored in an "extended ACL" that is essentially like the ACL of pre-SR10 releases.

New and Changed Rights

Mapping between pre-SR10 and SR10 rights is discussed completely in Chapters 2 and 3.

The keep (**k**) right, which is new at SR10, provides a way to protect an object from deletion even though it is cataloged in a writable directory. Attempts to **unlink**(2) an object with the **k** attribute return EACCESS, just as though the containing directory had no write permission.

In addition to being undeletable, an object with the **k** bit set has its name protected from change. Before SR10, it was not possible to protect a single object in a directory in this fashion, without protecting all objects in the directory from name changes.

Attempts to delete an object by using the following Aegis system calls

name_$delete_file
name_$delete_fileu

succeed only if the containing directory is writable and the file does not have a **k** attribute.

The Aegis system calls

name_$delete_file_force
name_$delete_fileu_fwu

forcibly delete an object with the **k** attribute if the caller has **p** rights to that object.

Summary of Incompatibilities

The SR10 changes in the ACL manager introduce several incompatibilities for programs. These incompatibilities are most pronounced in mixed networks of SR10 and SR9.x systems.

Incompatible System Interfaces

The incompatibilities engendered by changes in the ACL manager are small, limited in general to programs that examine individual rights bits.

At SR10, passing in old (supported at SR9.x but not at SR10) rights does not generate an error. But programs that examine rights bits need to be modified if they expect to see any bits from the set **gndcale** or do not expect to see **k** rights.

The **acl** and **edacl** commands support a different set of rights specifications. Shell scripts that examine the output of these commands may need to be modified.

As stated in earlier sections of this guide, it is not possible to modify any subset of **cale** rights on an SR10 directory. If, for example, an SR9.x node attempts to set **cl** rights on an SR10 directory, no rights are changed. Only programs that modify all or none of the **cale** set get exactly what they ask for.

When SR9.x tries to access an SR10 directory, it is allowed either all modify operations on that directory (the SR10 ACL includes **w**, which SR9.x sees as **cale**), or no modify operations (SR10 ACL has no **w**, so SR9.x sees none of **cale**). If the **k** right is set on an SR10 ACL, no **d** right is seen by an SR9.x node.

Recommendations for Data Alignment

Since current and future hardware and software releases penalize programs that make use of data structures in which data is not naturally aligned, we urge programmers to use "naturally" aligned data structures in their programs.

The following recommendations regarding data alignment are intended for programmers who are developing new applications, or who want to be sure their existing applications take full advantage of future hardware and software technology. Any programs intended to run on Apollo's Series 10000 machines should employ natural alignment.

Natural Alignment

A value that begins at an address that is a multiple of its size in bytes is said to be "naturally" aligned. For example, a 2-byte value is naturally aligned if it starts on a 2-byte address boundary. Similarly, an 8-byte value is naturally aligned when it begins on an 8-byte boundary.

Natural alignment trades a possible inefficiency in memory usage for an assured increase in processing speed. Many modern processors, such as the MC68020, are designed to transfer data most efficiently if the data is naturally aligned.

We expect future Apollo products to show a significant loss in performance when operating on data that is not naturally aligned. While SR10 versions of the Domain compilers allocate scalar variables on natural alignment boundaries where possible, the way in which you define structured data types such as records and arrays of records affects the alignment of data within the structure.

Padding Structured Data

By default, current and future Domain compilers only enforce the alignment of data within structures on byte or word boundaries; double-word and quad-word quantities are only word aligned. As a result, data within structured types will not always be naturally aligned unless you design the structure with such alignment in mind or you change the default alignment setting.

For example, the following Pascal TYPE declaration specifies a record that consists of a 2-byte integer followed by a double-precision floating-point number:

```
TYPE myrec = RECORD
field1: INTEGER16;
field2: DOUBLE;
END;
```

In this structure, a floating-point number (DOUBLE) immediately follows an integer (INTEGER16), which means that the DOUBLE ends up aligned unnaturally (at an address that is a multiple of 2, rather than 8). The following figure shows how this structure is written into memory:

INTEGER		DOUBLE							
####	####	####	####	####	####	####	####	####	####
0	1	2	3	4	5	6	7	8	9

In a simple structure like this one, you can get natural alignment by simply changing the order of the fields, so that the larger field is first, as shown:

```
TYPE myrec_new = RECORD
    field1: DOUBLE;
    field2: INTEGER16;
END;
```

The structure now looks like this:

DOUBLE								INTEGER	
####	####	####	####	####	####	####	####	####	####
0	1	2	3	4	5	6	7	8	9

You can always get natural alignment in a structure that contains only unstructured types by arranging the record in order of decreasing field size, as we have done in the **myrec_new** example.

Such a strategy cannot always work with more complex data structures (that is, structures that include other structures). For example, rearrangement of fields does not work with an array of **myrec** or **myrec_new** records. An array of **myrec_new** looks like this:

DOUBLE			INTEGER		DOUBLE			INTEGER	
####	#..#	####	####	####	#..#	####	####	####	####
0	1..6	7	8	9	A	B..10	11	12	13

Here, **myrec_new[2].field2** starts on address A, which is not an 8-byte multiple. Because the **myrec_new** record is not an even multiple of eight bytes, many of the **field2** fields in the array will not be naturally aligned.

In this case, you need to pad the record out until it becomes a multiple of eight bytes (the size of an INTEGER16), as shown below:

```
TYPE myrec_for_array = RECORD
    field1: DOUBLE;
    field2: INTEGER16;
    pad: array[1..3] of INTEGER16;
END;
```

As a general rule, you can get natural alignment in an array of structured types by padding the structured type to the size (or a multiple of the size) of the largest unstructured type. Similar techniques work with records that contain structures and other complex structured types.

Summary of Recommendations

We recommend that programmers take the following steps to ensure the best performance for their programs:

- Make sure that all new data structures follow natural alignment rules.
- Make sure routines that return pointers to storage (for example, free storage allocators) point to naturally aligned storage.

The SR10 version of the Domain Pascal compiler provides additional techniques for enforcing natural alignment through the use of attribute extensions. Since these extensions will not be supported on other systems, developers interested in portability may wish to design data structures that achieve natural alignment through padding.

Dot and Dot-Dot

At SR10, we extended our support for the hard links "dot" (this directory) and "dot-dot" (the parent of this directory). These directory entries cannot be deleted.

Summary of Changes

At SR10, the hard links "." (dot) and ".." (dot-dot) appear as the first two entries in any SR10 directory opened by using the "dir" calls (see "Changed Interfaces" next). In addition, the following new calls see these entries:

name_$read_diru_lc
ios_dir_$readdir

Programs must tolerate both the absence or presence of dot and dot-dot, and not assume that they are the first two entries in a directory.

Changed Interfaces

At SR10, the following UNIX system calls now return dot and dot-dot as the first two entries in a directory:

opendir
readdir
scandir
seekdir
telldir

The behavior of these interfaces remains otherwise unchanged.

At SR10, the **bsd4.3** version of **ls**(1) returns the names "." and ".." when invoked with the **-a** option, as does the Aegis **/com/ld** command invoked with the **-h** option.

New Interfaces

The following new Aegis interfaces all return dot and dot-dot as the first two entries in a directory.

name_$read_dir_lc	Reads the contents of a directory. First two entries in SR10 directories are always "." and "..". Names are returned in the correct case. Directory is specified by pathname.
name_$read_diru_lc	Reads the contents of a directory. First two entries in SR10 directories are always "." and "..". Names are returned in the correct case. Directory is specified by UID.
ios_dir_$readdir	Reads the contents of a directory. First two entries in SR10 directories are always "." and "..". Names are returned in the correct case. Directory is specified by **ios_id**.

All Apollo programs that read directories have been modified to use these new interfaces.

Summary of Incompatibilities

We do not expect the appearance of dot and dot-dot to cause problems for any pre-SR10 software. In general, UNIX programs expect these entries and do not attempt to delete them.

Any program that opens a directory by using one of the "Changed Interfaces" previously listed and that does not recognize dot and dot-dot as required contents of every directory (that is, a program that attempts to delete these entries, or does not consider a directory empty when it contains only dot and dot-dot) fails at SR10.

It is not enough to have a program ignore the first two directory entries, since dot and dot-dot were not returned by pre-SR10 releases, and the first two entries would be something else in an SR9.x directory or in a foreign file system. Nor, for the same reasons, is it correct to assume that any directory that only shows two entries is, in fact, empty.

Changes to Data Structures

At SR10, we've merged the **bsd4.3** and **sys5.3** versions of certain data structures and increased them in size to allow for future extensions. The data structures affected are

- The structure returned by the **stat, lstat,** and **fstat** system calls (declared in **<sys/stat.h>**)
- The structures returned by the **opendir, readdir, getdents** system calls (declared in **<sys/dir.h>** for **bsd4.3** and in **<dirent.h>** and **<sys/dirent.h>** for **sys5.3**

The implications of this change are twofold. If you have programs that contain your own declarations of these structures, the programs probably will not work any longer. Also, you'll have to recompile a program to change its SYSTYPE from **bsd4.2** to **bsd4.3** (or from **sys5** to **sys5.3**).

SYSTYPEs

SR10 provides no compile-time or run-time support for 4.1BSD (SYSTYPE=**bsd4.1**) or AT&T System III (SYSTYPE=**sys3**). Programs that explicitly declare a SYSTYPE of **sys3** or **bsd4.1** are rejected by the loader. The following error appears when an object module has the SYSTYPE of **sys3**:

```
object module systype sys3 obsolete;
see sr10 release notes (process manager/loader)
```

Any **bsd4.1** modules generate the following error:

```
object module requires unsupported systype
(process manager/loader)
```

Type Managers and SYSTYPEs

A type manager runs with the system type of the program that invokes it. At SR10, possible SYSTYPEs include **bsd4.3**, **sys5.3**, **bsd4.2**, and **sys5**. If you have written type managers that use UNIX system calls, you should verify that none of these calls is SYSTYPE-dependent. If you are not sure, we recommend that you test the type manager with all the possible SYSTYPEs.

Incompatibilities between System V Release 2 and 3

Incompatibilities between Release 2 and Release 3 are documented in the standard documentation for AT&T System V Release 3.

Incompatibilities between 4.2BSD and 4.3BSD

Incompatibilities and changes between 4.2BSD and 4.3BSD are documented in the 4.3BSD *System Manager's Manual*, Chapters 12 and 13.

Some New Features of the Domain C Compiler

The following sections highlight some of the features available with the SR10 Domain/C Compiler. Consult the *Domain C Language Reference* (order number 002093) for more information about these and other compiler-related features.

Function Prototypes

Function prototypes allow function declarations to include data type information about arguments and are the default at SR10. If you compile a source program that contains an old-style declaration or invokes a function before the prototype, compile the program with the **-ntype** option.

By default, the compiler is silent about old-style declarations. However, if you use the **-info** option when you compile, the compiler prints informational messages.

Informational Messages

The Domain/C compiler can now issue informational messages in addition to warning and error messages. Information messages identify portions of the source file that, if rewritten, could be more efficient or portable. The new **-info** option instructs the compiler to produce informational messages; without it, the compiler produces only warning and error messages.

Run-Time Version Specification

With the **-runtype** *systype* option, you can compile a C program that contains one SYSTYPE setting (such as **sys5.3** or **bsd4.3** and then run the program in another environment.

The Section Specifier

The **/com/cc** command creates a named section in an object file for every global variable declared; on the other hand, **/bin/cc** puts all initialized global variables in one section, and all uninitialized global declarations in another. To make **/bin/cc** behave like **/com/cc**, you can now use section specifiers to create named sections. Section specifiers are especially useful for interacting with FORTRAN programs that use common blocks.

Reference Variables

Reference variables are variables that refer to other objects. Whenever a reference variable appears in an expression, the referenced object is accessed instead. With a reference variable, you can create an *alias* for a variable, transform a constant into a form of variable, and provide a clean syntax for passing function arguments by reference.

Built-In Function

Domain/C supports built-in functions for all routines declared in **<float.h>**, **<string.h>**, and **<strings.h>**. To use a built-in function, which can significantly increase execution speed, you must include the **<builtins.h>** header file in your program.

Disk Storage: malloc() and rws_$alloc

In order to guarantee that backing store is available for the virtual memory allocated, the calls **malloc()** and **rws_$alloc** now attempt to reserve the specified amount of disk space. If the requested amount of disk space isn't available, then the calls fail. This means that free disk space is reserved to programs that **malloc** large areas even if the area is never used. Disk space is freed when the program exits, or if the space is explicitly released with **free()** or **rws_$release_heap**.

Disk Storage and Static Arrays

Before SR10, a program could declare a large static array and the disk space would only be consumed as the array filled it in. At SR10, all of the disk space that an array requires is reserved when the program is loaded. This fact may cause a program to require more disk space to run than it did before.

Some New Features of the Pascal Compiler

The Domain Pascal compiler generates COFF object modules. If you need to compile into the pre-SR10 obj format, use the SR9.7 compiler.

Preprocessor Variable

A new preprocessor variable, _BFMT__COFF, is available; it is both defined and enabled. With this variable, you can determine whether the compiler you are using is a COFF compiler.

Syntax for Specifying Size and Alignment

Domain Pascal has a new syntax for specifying size and alignment of objects. See the *Domain Pascal Language Reference* (order number 000792) for details.

Routine Signatures

In the past, if a routine was declared FORWARD or EXTERN and the body of the routine was declared later in a source file, it was illegal to respecify the routine's parameter and routine option lists. The compiler now accepts a redeclaration and issues an error if the declarations are not equivalent.

Signature Comparison Standards

Standards for procedure signature comparison when assigning the address of an actual routine to a procedure or function pointer are more stringent at SR10. Previously, parameters were only required to be assignment compatible. At SR10, parameters must be name compatible. Under the old rules, it was possible to have radically different declarations that were considered compatible. (For example, the unsigned subrange 0..65535 was considered compatible with the signed range, INTEGER.)

Some New Features of the FORTRAN Compiler

The following headings highlight some of the features available with the SR10 Domain FORTRAN Compiler. Consult the *Domain FORTRAN Language Reference* (order number 000530) for more information about these and other compiler-related features.

- Compatibility with UNIX Formatted Binary-File Format
- Strings Delimited by Single (') and Double (") Quotes
- Comma (,) Now Available as Input Delimiter

Data Type COMPLEX*16

At SR10, we have added a double-precision version of the COMPLEX data type called COMPLEX*16. You can declare variables by using the COMPLEX*16 command or the DOUBLE COMPLEX command.

New functions have been added to the FORTRAN library to support the COMPLEX*16 type.

I/O to Streams

You can now attach a stream opened with the **ios_$open** system call to a FORTRAN unit and perform I/O to that stream.

INCLUDE Syntax of UNIX f77 Compiler

You can now use the same syntax as the UNIX f77 compiler to INCLUDE external source files. For example, the statement **INCLUDE 'mysource.ins.ftn'** includes the same file as **%INCLUDE 'mysource.ins.ftn'**. The new syntax, however, must comply with the same restrictions that apply to any other Domain FORTRAN statement.

Compiler Options

The **-uc** (UNIX Compatibility) option controls the following features. The option's default is OFF.

Appended Underscore (_)	This feature adds an underscore (_) to the names of SUBROUTINE, FUNCTION, or COMMON block names which do not begin or end with an underscore (_) and which lack a dollar sign in their names.
Default File Names	This feature automatically names the file **fort.<unit_number>** if you don't provide a FILE= specifier in the OPEN statement. If the **-uc** option is OFF, the file takes the name **for<unit_number>.dat**.
UNIX special characters	At SR10, the compiler recognizes UNIX special characters (for example, \n) when you compile a program with the **-uc** option.

Uppercase Option: -u

The **-u** option leaves the case of characters in identifiers as entered; by default, the compiler converts the characters in identifiers to lowercase characters.

Free Format Option: -ff

The **-ff** option allows source lines to be a maximum of 1024 characters long; an ampersand (&) in column 1 specifies a continuation line. The option's default is OFF.

Tabs

A tab character in any of the first six columns causes a skip to column 7. In all other columns, it's treated like a space character.

Unit 0

By default, UNIT 0 connects directly to **-stderr**.

Flexnames

Prior to SR10, variable names were only unique through the first 32 characters. That is, the compilers interpreted the following two strings as the same identifier because the first 32 characters are the same.

```
this_is_a_very_long_and_annoying_name
this_is_a_very_long_and_annoying_identifier
```

At SR10, these strings are interpreted as distinct identifiers. Identifier names can now be up to 4096 characters in length.

Support for f77 Commands and Options

Domain systems now support the **f77** command, which enables you to invoke the Domain FORTRAN compiler with UNIX command options. Note that if you invoke Domain FORTRAN with **f77**, the compiler employs **f77** semantics. If you compile with the **ftn** command, the compiler provides Aegis semantics.

New Debugger

SR10 replaces the **/com/debug** command with a new high-level language debugger, Domain/DDE, which can debug both obj objects produced by SR9.5 and higher release compilers and COFF format objects produced by SR10 compilers. It is installed as part of the base software. The **dde** command resides in the **/usr/apollo/bin** directory. Extensive online help is provided, as well.

Domain/DDE provides many new features, including a graphic, menu-driven interface with language-sensitive expression evaluation, multiprocess debugging, and cross-node debugging. The new debugger also has macros to provide command-line syntax compatibility with old **debug** and **dbx** commands. Domain/DDE cannot debug programs built before SR9.5.

Compatibility with Domain/Debug (/com/debug)

As a transition aid, we have included macros to simulate Domain/Debug syntax in the directory **/sys/debug/old_debug_macros**. You can copy the macros to your own startup file (**.dderc**) and modify them, or load them directly by typing the following:

```
input -from /sys/debug/old_debug_macros
```

For equivalents to Domain/Debug debugger variables (that is, **set `<debugger_variable> = <value>**), see the "property" commands.

Precautions for Mixed Networks

Some additional precautions are necessary for programming and running programs in mixed networks.

Links

Be careful when creating links from nodes running pre-SR10 software to those running SR10 (COFF) software. Even though such links may have worked fine in the past, nodes running pre-SR10 software cannot execute SR10 object files.

Command Search Rules (PATH)

Similarly, command search rules often specify your current working directory in the first position. Problems therefore arise if you set your working directory to an SR10 directory containing executable files. For example, if you are working on an SR9.x node and you try to change working directory to an SR10 node directory that contains binaries, and you attempt to execute one of them, the attempt fails since you cannot execute COFF objects on pre-SR10 nodes.

Type Managers

Do not compile type managers as COFF objects, since type managers sometimes get passed between nodes and may therefore find their way to pre-SR10 nodes. Should this happen, objects of the affected type(s) would not function on the pre-SR10 nodes.

The C Compiler

Note that, even though you can delete an SR10 COFF file from an SR9.x node, invoking the C compiler on the pre-SR10 node does not delete an existing binary file. This is because the SR9.x compiler only operates on obj format files. If you were able to do this, you would be replacing a COFF file with an obj file, which is unlikely to be what you intended.

SR10 and the Domain Software Engineering Environment

If you are a DSEE user, there are several important considerations that you must take into account in making the transition to SR10. We outline them here. For further details on the issues discussed in this section,

please see the release documents for DSEE Version 3.3 (the SR10-compatible version of the software) and DSEE Version 3.2 (the SR9.7-compatible version of the software that can share DSEE objects with DSEE Version 3.3). Each of these documents is online in the **/install/doc/apollo** directory of the DSEE software shipment.

Colons in Element Names

As a consequence of the case sensitivity introduced in SR10, you must rename existing elements whose names contain colons if these elements are to reside on SR10 nodes. Renaming is necessary because the element names themselves are interpreted as containing uppercase characters (for example, :a is interpreted as A), while the DSEE library database still retains the colons in its interpretations of the element names. As a result, it is impossible to access history on the elements and, perhaps, even the elements themselves.

We recommend that you do this renaming *before* you install SR10 on a node. Below is a simple procedure that you can run to rename elements with colons in their names. Do this for each library on the node that will run SR10.

```
DSEE> sho ele ?*:?* -format "ren ele %ele %ele" > colon.dsee
# Edit the colon.dsee file to remove colon(s) from new name
# (the name on the right)
DSEE> <colon.dsee
```

Once you've installed both SR10 and DSEE Version 3.3 on the node, you can rename the elements again, putting colons or uppercase characters in the names, as you wish. (Note, however, that colons are interpreted differently on SR10 nodes and SR9.7 nodes. If you give elements residing on SR10 nodes names that contain colons, you will be unable to access those elements from DSEE Version 3.2. See the DSEE Version 3.3 Release Document for more information.)

If you don't rename the elements before you install SR10 and DSEE Version 3.3, you have to perform two other steps before you can rename the elements whose names contain colons. Below is an example of these steps:

```
# Delete the stub files that the DSEE history manager uses
# (You must be an administrator of the library to do this)
$ dlf //sr10_node/dsee_3.3_lib/?* -f
# From within the DSEE environment, recover the library
DSEE> recover library -full
```

Once the library has been recovered, those DSEE elements whose names contained colons prior to SR10 will still have colons in their names, and the naming will be consistent between element names and the library database. You can rename the elements with uppercase characters in the name, if you choose.

Bound Configuration Threads (BCTs)

Because of a bug fixed in the way the configuration manager stored BCTs in pools, pool-stored BCTs generated with versions of DSEE software prior to Version 3.2 (or with beta test versions of either DSEE Version 3.2 or DSEE Version 3.3) cannot be reused with this release of DSEE Version 3.2 or DSEE Version 3.3.

BCTs stored in release areas, however, are reusable. If you have one or more existing component builds whose BCTs you want to reuse, we recommend that you place the BCTs in release areas before you install this release of DSEE on your node.

Copying DSEE Objects with cp and cpt

Because installing SR10 requires that you **invol** your disk, you will want to make copies of all your DSEE objects, either on tape, on another medium, or on another node's disk, prior to using **invol**.

If you choose to use either the BSD and SysV command **cp** or the Aegis command **cpt** to copy DSEE objects to another node, you must ensure that copying preserves subsystem seals. When using **cp**, include the **-P** option and the **-o** option on the command line. The **cpt** command preserves subsystem seals by default; do not override this behavior by issuing the command with either the **-nsubs** option or the **-dacl** option.

Case Sensitivity and DSEE

Once you have installed SR10 and DSEE Version 3.3 on a node, you must ensure that all references to pathnames in system models and source code are case sensitive. Please see the DSEE Version 3.3 Release Document for more details.

Running in a Mixed Environment

If you run DSEE software in a network that contains nodes running more than one release of the operating system, you must give attention to the items described in the following subsections.

Sharing DSEE objects

Unlike prior releases of DSEE software, DSEE Version 3.3, the SR10-compatible version of the software, will *not* be able to share DSEE objects in a mixed network of pre-SR10 nodes running earlier releases of DSEE software. The one exception to this rule, of course, is DSEE Version 3.2, a version of the software that runs on SR9.7 nodes, which was built to be compatible with DSEE Version 3.3. Except under the conditions described later in this section, DSEE Versions 3.2 and 3.3, and compatible subsequent releases of DSEE software, can access the same libraries, elements, and systems.

Distributed Building in a Mixed Network

The DSEE **set builder** command enables you to execute component builds on one or more remote nodes. DSEE Version 3.2 and Version 3.3 both can invoke builds on nodes running SR9.7 and SR10. However, some portions of your translation rules may be operating-system-release dependent: commands that are executable on an SR9.7 node may not be executable on an SR10 node, and commands that are executable on an SR10 may not be executable on an SR9.7 node.

The most important factor in determining the success or failure of a translation rule in a mixed network is the set of translators resident on the reference node (identified in the **-reference** option to the **set builder** command). If you are using a mixed set of SR9.7 and SR10 nodes as build servers, it's generally best to use a node running SR9.7 as your reference node; SR9.7 programs are more likely executable on both SR9.7 and SR10 nodes than are SR10 programs. (If it's important that you use SR10

translators to build your system, we advise that you use only nodes running SR10 as builder nodes and that you identify an SR10 node as the reference node.)

The following table lists some translators and shells and their ability to be executed on SR9.7 nodes. This list is not complete.

Program	SR10 runs on SR9.7?	SR9.7 runs on SR10?
# Aegis shell		
/com/sh	yes	yes
# bsd4.2/3 shells		
/bin/sh	no (COFF object)	yes (obj object)
/bin/csh	no (COFF object)	yes (obj object)
# sys5 shells		
/bin/sh	no (COFF object)	yes (obj object)
/bin/csh	no (COFF object)	yes (obj object)
/bin/ksh	no (COFF object)	n/a (no SR9.7)
# Apollo Compilers		
/com/pas (obj output)	yes	yes
/com/pas.coff (COFF output)	no	n/a (no SR9.7)
/com/ftn (obj output)	yes	yes
/com/ftn.coff (COFF output)	no	n/a (no SR9.7)
/com/cc (obj output)	yes	yes
/com/cc.coff (COFF output)	no	n/a (no SR9.7)
/com/asm (COFF output)	yes	n/a (no SR9.7)
# Apollo Binder		
/com/bind (COFF input only)	no	n/a (no SR9.7)
/com/bind (obj input only)	yes	yes
# UNIX Compilers, Linker		
/bin/cc	no (COFF object)	yes (with /usr/lib/cc, /usr/lib/bind)
/bin/ld	no (COFF object)	yes
/bin/ar	no (COFF object)	yes
# Others		
/usr/bin/lex	no (COFF object)	yes
/usr/bin/yacc	no (COFF object)	yes
/usr/bin/m4	no (COFF object)	yes

Try executing the program to determine if a particular program not listed here can be executed by a remote node running SR9.7 or SR10. (Executable code is shipped in obj format except where noted.)

Note also that type managers may change from release to release, and that they are always loaded via the reference path. For example, if the reference path points to an SR9.7 node and DSEE is building on an SR10 node, then unexpected errors may occur.

Note to Pascal Users

If your DSEE system models employ the **make_visible** declaration, then you should use the SR9.7-compatibility mode Pascal compiler. Previous versions of this compiler had a bug that would cause objects referenced by **make_visible** to be lost.

Case Sensitivity in a Mixed Environment

DSEE Version 3.3 with SR10 and later releases is case sensitive; however, because SR9.7 is case insensitive, once you start using mixed-case or uppercase letters in the names of DSEE objects residing on SR10 nodes, you will be unable to access those objects from an SR9.7 node (either running DSEE Version 3.2 or acting as a build server). SR9.7 nodes using DSEE to attempt to resolve pathnames containing uppercase letters for any DSEE objects (for example, system directories, elements, etc.) will fail. (This would also be true for SR10 nodes and DSEE Version 3.3 if you set the DOWNCASE environment variable to **true**.)

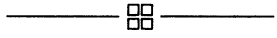
DSEE Object Names with Hyphens and Plus Signs

With DSEE Version 3.3, you can assign names containing plus signs (+) or hyphens (-) to libraries, elements, branches, versions, and logical pools. However, if you do, you will be unable to access those objects from DSEE Version 3.2.

Building COFF vs. obj-Derived Objects

You may want to write your system model in such a way that it can generate either COFF format derived objects or obj format derived objects. This is easy to do, particularly if you use the **alias** declaration in your system model to define aliases for translators, and make the definitions

conditional upon **-target** predicates identified in your model thread. Please see the DSEE Version 3.3 Release Document for an example of system model code that would produce either COFF-derived or obj-derived objects.



Appendix A

Protection and ACLs

The *Managing System Software* manuals provide a detailed view of protection and Access Control Lists (ACLs) at SR10, and several sections in this manual describe changes made to ACLs and protections. This appendix describes the differences between the SR10 protection model and both the standard UNIX software model and the pre-SR10 models, as well as their implications for users converting to SR10.

The following sections discuss the SR10 protection model in the context of both UNIX software and pre-SR10 Domain software, and describe some of the transition issues. See the *Managing System Software* manuals for a detailed description of protections and ACLs at SR10, and the tools that you use to manage them. Also see *Installing Software with Apollo's Release and Installation Tools*, and particularly Chapter 6 of that book, for information on how to manage protection when you install SR10-based software.

Overall Network Protection Model

The Apollo system protection model allows each installation to determine the best way to organize the protection of files and directories for that installation. The model supports the full range of protection choices, from totally open to tightly protected, and anywhere in between.

Some installations give node owners full control over their nodes, including the system software. In an open network, all the files on a node's disk

"belong" to the owner of the node, who is free to protect, delete, or move the files at his discretion. (There are exceptions, of course. Certain system files must not be deleted or moved, or else the system will not operate correctly or will be open to intrusions. In the UNIX environments, certain directories must be controlled, since many procedures that run as root use programs from these directories, and the susceptibility to some security problems is great.) However, within this framework, the node owner in an open network may tailor his node to meet his needs.

On other networks, nodes are considered a group resource, and are administered by a central group and tightly controlled. In a protected network, all system files on a node's disk "belong" to the system administrator and are protected against modification by users of the node. Protections are maintained so that only the system administrator can alter files not specifically private to a user.

At SR10, software on such nodes is owned and generally controlled by **root (locksmith)**, rather than by **sys_admin** as in previous releases. Protected installations, especially those that run in the Aegis environment, may want to migrate their protections from control by **sys_admin** accounts to control by **root** accounts.

The UNIX Protection Model

The UNIX system protects objects with three rights: read, write and execute (**rwx**). For a directory, the execute right is the right to search through the directory to resolve pathnames. The UNIX system recognizes three subject classes for protection: user, group, and "other" (Aegis terminology refers to "user" as owner and "other" as world). Rights are granted to the owner of the object, a group associated with the object (and therefore people with accounts in that group), and the rest of the world. These rights are generally represented as **rwxrwxrwx**, with the leftmost **rwx** referring to the user rights, the middle **rwx** referring to the group rights, and the rightmost **rwx** referring to the "other" rights.

Protection for newly created objects is derived as follows: The user identity is obtained from the user-name identity of the creating process. The group identity is obtained from the group-name identity of the creating process (for SysV) or the group associated with the containing directory (for BSD). The object's initial protection is derived from the set of rights specified by the creating program and a mask, known as the umask, that is associated with the creating process.

The umask is a 9-bit mask in the standard "rwxrwxrwx" form that specifies rights that are to be automatically subtracted from all newly created files and directories. For example, if a file is created with rights of "rw-rw-rw-" and the umask is "----w--w-", the file will get rights of "rw-r--r--." A typical use of the umask might be to deny certain rights to "other" so that files that are created by applications running on behalf of the user will not, by default, be writable by all others. The value of the umask is set by the **umask** command.

SR10 Extensions to the UNIX Model

The SR10 protection implementation extends the UNIX model by enabling you to specify additional rights, organizational divisions, and subject categories. The following sections describe these extensions in detail.

Extended Access Rights

We have extended the rights from **rwx** to **prwxk**, where **p** specifies complete control over the object (that is, the right to modify its protections), and **k** specifies that an object may not be deleted, even in an otherwise writable directory.

Extended Organizational Divisions

We have extended the number of recognized organizational divisions from two (person and group) to three (person, group, and organization). This more closely matches the real world divisions that occur in large organizations and large networks. As a result, we have expanded the number of subject categories that can be specified for protection purposes from three (user, group, and other) to include an organization entry. However, we have provided a mechanism for marking an object so that the organization identifier is ignored during access checking, so that the pure UNIX behavior can be maintained. Note that the registry can use the organization identifier as a basis for control over user accounts. When you make the transition to SR10, and if you want to have a pure UNIX environment, you should make sure that the registry also does not use organizations.

Extended Subject Categories

We have extended the number of subject categories for protection to allow you to specify additional subjects in an ACL. Each subject so named is referred to by a triplet Subject Identifier (SID), consisting of a person, group and organization. SID's are specifiable exactly or with wildcards for any of the three fields (for example, **joe.os.r_d** or **%.%.mktg**). The ACL also contains the rights granted or denied to the SIDs named in the list.

Benefits of Extensions

These extensions

- Allow groups of people to exercise joint control over sets of files, since several people may have **p** rights to the files
- Allow selected objects in a directory to be protected, through the **k** right, without restricting the protection of other objects coresident in the directory
- Allow different access rights for specific listed subjects than for the rest of the world, including increased or lessened privileges

Protection Inheritance Enhancements

Protection inheritance has also been enhanced. In addition to supporting the System V and BSD semantics for protecting newly created objects, every Domain/OS directory has two initial default ACLs, one for files and one for directories, in addition to the ACL that protects the directory itself. At the user's discretion, these initial default ACLs, rather than the UNIX inheritance rules, can be used to control the initial protection values for files and directories created within the directory.

The inheritance that can be specified in an initial default ACL is very flexible. The user can independently specify whether the user, the user rights, the group, the group rights, the organization, the organization rights, or the "other" rights get inherited from the initial ACL or from the creating process. Extended ACLs are also supported for initial default ACLs.

The UNIX umask is also supported in the Aegis environment, and a **umask** command is provided for the Aegis shell. This is a UNIX system compatible command, so that only **rwx** rights for person, group, and other, but not organization, can be specified. (Note that files created by some other process on behalf of the first process, for example by the Display Manager, do not get the benefit of the first process' umask.)

Interactions of UNIX Protection and ACLs

The standard UNIX system calls and commands used for setting or interrogating protection capabilities are not extensible and cannot support the additional information in an ACL. However, when one of these system calls or commands is used, its behavior may, nevertheless, affect or be affected by the extended protection information. This section describes the interactions between UNIX protection and ACLs.

Extended Entry Rights Mask

To control this interaction, each file's ACL has an extended entry rights mask, whose bits correspond to **prwxk**. When rights are granted by an extended ACL entry, the rights in the ACL entry are first ANDed with the extended entry rights mask before they are used for access determination. Thus, if the extended entry rights mask specified "-r---," only read rights are granted via an extended ACL. The use of a mask to limit permissions is required so that UNIX system calls and commands perform as expected, regardless of the presence of an extended ACL.

Initially, the extended entry rights mask is the OR of all the rights in the extended entries in the ACL, thus allowing an extended ACL to grant all rights it contains. The subsequent value of the mask is determined by the **chmod**() system call.

The Effects on the Mask by the chmod() System Call

At SR10.0 and after, when a **chmod**() system call is issued

- The "other" rights specified in that call becomes the new value of the extended entry rights mask.

- Any **p** and **k** rights are turned off in the group and world entries and in the extended entry rights mask.
- Any **k** rights are turned off in the user entry.
- The organization entry is marked as ignored.

Thus when **chmod()** sets a value for "other" rights, it is in effect saying that the *maximum* rights to be granted to anyone other than user or group are the "other" rights. If a given SID matches an extended ACL entry, the only possible effect of the extended ACL entry is to *reduce* that SID's rights to less than those that would be granted via "other" rights.

At the same time, the **chmod()** call removes the effects of the other Domain/OS extensions to the UNIX protection mechanism; that is, the special granting of **p** and **k** rights and the organization entry.

The Effects on the Mask by the chmod Command

The UNIX **chmod** command, which uses the **chmod()** system call, supports two distinct forms. The first form sets the access rights to an absolute value; the second adds or removes specific parts of the existing access rights. The effect of these two forms on the extended entry rights mask is as follows.

Form 1: The first form, illustrated by **chmod 664**, sets the "other" rights. In this case "4" (or bits "100"), is used as the middle three bits (rwx) of the extended entry rights mask. Also, as described previously, **chmod** always sets the **p** and **k** bits, which are not supported by the UNIX system, to 0. So in this example, even if the mask was initially "prwxk," it is now "-r---." Note that the extended ACL entries are not discarded, even if you specify **chmod** *xx***0**. The list of entries is maintained but is rendered ineffective.

Form 2: The second form, illustrated by **chmod g–r** or **chmod g+r**, is used to change rights based on current protection values (rwx). For example, **chmod g–r** specifies that the read rights are to be removed from the group entry. As described in the previous section, this form of the **chmod** command also sets the **p** and **k** bits to 0. And the rwx rights in the extended entry rights mask are changed (even if the rights were not specified by the command).

Implications for the stat() System Call

Similar considerations apply to reporting the protections in effect for a file. The **stat**() system call is the standard way for a UNIX program to determine the access rights that apply to a file. Because this system call is well established, it cannot be changed to report the extended ACL rights. However, it is important that the system never underrepresent the rights that are in effect for a given file. Therefore, the value of "other" rights reported by the **stat**() system call includes the ORing of the following:

- The actual world rights
- The organization rights, if they are not marked as ignored
- The extended entry rights mask

For example, if

- The actual world rights are "-r---"
- The organization rights are "---x-"
- The extended entry rights mask was "-rw--"

the value reported by the **stat**() system call would be (ignoring **p** and **k** which are not returned):

```
     'r--' OR '--x' OR 'rw-'
=    'rwx'
```

NOTE: If any Domain/OS extensions to UNIX protections are in effect for a particular object, an **ls -l** command will show a plus sign (+) identifier at the end of the rights string; for example, "-rwxr-x--x+."

Apollo's Extended UNIX Commands

Although the standard UNIX commands do reasonable things to extended ACLs, you can only get the full benefit of the extended protection features in a UNIX environment, if you use the extended UNIX commands provided by Apollo (**chacl, lsacl,** and **cpacl**). The standard UNIX commands cannot be coerced into retaining all aspects of the new protection

semantics. Also, note that UNIX applications that use the **chmod()** system call will affect the extended protection on any files or directories to which they apply **chmod()**. The effect may be to limit the rights granted, or it may disable the extended ACL or organization rights altogether. It will, however, never grant *more* rights to an entry listed in an ACL than is specified in "other" rights.

ACL Search Order

This section describes the matching rules that are used for access checking. In the description, a percent sign (%) is the wildcard specifier in an ACL entry and *[xxxx]* indicates all possible values for that field, including the wildcard.

To understand how the system determines access rights, you must understand the search order within an ACL. ACL entries are always searched from the most specific to the least. For example, entries with a person name are always searched before entries with a wildcard (%) in the person field. Entries of the form *person.group.%* are searched before entries of the form *person.%.%* since the allowed group is exactly specified.

1. The "user" entry is checked first. If there is a match and the rights are not to be "ignored," that entry is used.

2. Extended ACL entries of the form *person.[xxxx].[xxxx]* are searched for an exact match in the person field and any other non-wildcarded fields. If a match is found, that entry is used after application of the extended entry rights mask.

3. The "group" entry is checked next. If there is a match, the rights are not to be "ignored," that entry is used.

4. Extended ACL entries of the form *%.group.[xxxx]* are searched for an exact match in the group field and any non-wildcarded organization field. If a match is found, that entry is used after application of the extended entry rights mask.

NOTE: In steps 3 and 4, if project lists are enabled, then each entry from the user's project list is substituted for his current project to form a SID. Each SID is then checked against the "group" entry (step 3) or against all extended ACL entries of the form *%.project.[xxxx]* (step 4). That is, the checks in steps 3 and 4 are repeated for each possible SID, including the effective SID. The rights from each match, if any, are concatenated (ORed) and used to determine access. (The project list is only created if either the PROJLIST environment variable is true or the SYSTYPE environment variable value is **BSD***xxx* (for example, BSD4.3) at login.)

5. The "organization" entry is checked next. If there is a match and the rights are not to be "ignored," that entry is used.

6. Extended ACL entries of the form *%.%.org* are searched for an exact match in the organization field. If a match is found, that entry is used, after application of the extended entry rights mask.

7. The "other" entry is used if no previous match was found.

Changes in Protections Between SR9.7 and SR10

The following subsections discuss changes that have been made in the way ACLs worked in SR9.7 and the way protection works in SR10. They also discuss some of the implications of these changes.

Identifiers

The node ID has been eliminated from the SID and from ACLs. As a result, you cannot control access by node ID, and SR10 nodes ignore any node ID values specified in pre-SR10 ACLs. However, we have added several mechanisms to limit access to local objects. These are discussed in the "Additional New Protection Capabilities" subsection at the end of this Appendix.

Required Entries and Ownership

At SR10, four protections entries are always present: user, group, organization, and other. These correspond to the standard UNIX protections, with the organization extension. As part of these entries, each object now has a user (owner) and is associated with a specific group and organization. Any member of the group or organization will be granted, at a minimum, the rights specified in the organization or group field.

When you copy a file from a pre-SR10 node to an SR10 system by using **cpf –sacl** to preserve the protections, the required entries are given the user, group, or organization ID of "none." The protection rights are marked "Ignore." Any rights specified for **%.%.%** are converted into the "other" required entry. The remaining SR9.7 ACL entries are simply converted into extended ACL entries. For example, a file ACL might look as follows on an SR9.7 node:

```
$ acl /misc/mail_names
Acl for /misc/mail_names:
  bill.%.%.%                          pgndwrx
  %.backup.%.%                        ----wr-
  %.%.r_d.%                           ----wrx
  %.%.%.%                             -----rx
```

If you use the **cpf –sacl** (on the SR10 node) to copy the file, you will then see the following protections for the new file:

```
$ cpf //h/misc/mail_names test2 –sacl
$ acl test2
Acl for test2:
Required entries
 none.%.%                            [ignored]
 %.none.%                            [ignored]
 %.%.none                            [ignored]
 %.%.%                               -r-x-
 Extended entry rights mask:         prwx-
Extended entries
 bill.%.%                            prwx-
 %.backup.%                          -rw--
 %.%.r_d                             -rwx-
```

You can use a **–conv[ert]** option to the SR10.1 (but not SR10.0 or pre-SR10) **cpf, cpt**, and **acl** commands to convert the SR9.7 ACL to SR10

required protections. This option sets the user entry in the target from the first *person.%.% entry* in the source with **p** rights. The first *%.group.% entry* in the source becomes the group entry in the target, and the first *%.%.org entry* in the source becomes the organization entry in the target. For example:

```
$ cpf //h/misc/mail_names test3 –sacl –conv
$ acl test3
Acl for test3:
Required entries
 bill.%.%                              prwx-
 %.backup.%                            -rw--
 %.%.r_d                               -rwx-
 %.%.%                                 -r-x-
 Extended entry rights mask:           -----
```

Note that **rbak** does not have a –**convert** option. However, if you use **rbak** to restore a tape that was generated by a pre-SR10 **wbak** onto an SR10 node, the –**convert** style conversion will be done automatically.

We have also provided a mechanism that enables SR10 protections to mimic pre-SR10 Aegis protections. You can mark any combination of the person, group, and organization required entries to be ignored. If you mark all three of the required entries this way, the protection extended ACLs behave in the pre-SR10 Aegis manner. You cannot mark the "other" entry (or extended ACLs) to be ignored.

You should also note the changes that we made for full UNIX compatibility, such as the extended entry rights mask, that are discussed in the previous section.

Inheritance Mechanism

Before SR10, Domain/IX allowed you to use UNIX right inheritance by having special initial directory and file ACLs. In this case both the "identities" and the permissions for new files and directories were inherited from the process (with BSD, the group ID was inherited from the containing directory). At SR10, this mechanism has been removed; the mode of protection inheritance is now coded directly in the initial file and initial directory ACL required entries. You can specify any possible combination of identity inheritance and rights inheritance for the required entries. (However, you cannot mark the extended ACL entries to inherit rights from the process.) For example, you could specify that all files created in

directory obtained their user, group, and organization identities and protections from the process, with an additional extended ACL entry that gives you pwrx rights.

In this case, **acl –if** would show something like:

$ acl test_dr –if

```
Initial (default) acl for files created under test_dr
Required entries                          For the current process:
 [from process] [specified by process]  bill.%.%    [specified by process]
 [from process] [specified by process]  %.none.%    [specified by process]
 [from process] [specified by process]  %.r_d.%     [specified by process]
 %.%.%          [specified by process]
 Extended entry rights mask:  prwx-
 Extended entries
  bill.%.%                    prwx-
```

Rights

At SR10 we have changed and simplified the protection rights as follows:

- We have eliminated **g** and **n** rights.
- We have combined the **c, a, l,** and **e** rights into a single directory **w** right.
- We have replaced the **d** with the **k** right. The **k** right is approximately the logical negation of the **d** right. An object with a **k** right cannot be deleted unless you are **root** or have **p** rights and use the **dlf –f** or **dlt –f** command. You can delete any object that does not have **k** rights if you have **w** rights for the directory.
- We have changed the **s** right identifier to **x**.

When accessing files or listing ACLs, SR9.7 automatically convert rights from SR10 format. SR10 nodes can convert protection rights from pre-SR10 format to SR10 format. A section in Chapter 3 of this manual, called "Protection Incompatibilities in Mixed Networks," describes how an SR10 node will see protections on SR9.7 nodes and vice versa.

NOTE: When you use the **cpf –sacl** command to copy from an SR9.7 node to an SR10 node, the exact results will differ depending upon whether you use the SR9.7 or SR10 version of the command. If you enter the **cpf** command on the SR9.7 node, **k** rights are set on the SR10 entries if the corresponding SR9.7 ACL entry did not have **d** rights set. If you enter the **cpf** command on the SR10 node, **k** rights are never set, independent of the status of the SR9.7 **d** right.

Tools for Manipulating Protections on Objects

In the Aegis environment, the tools are

edacl	Used to edit the protections of a specified object
acl	Used to display the protections on one or more objects and to copy protections from one object to one or more objects
ld –r	Used to display the current process' rights to listed objects

In the UNIX environments, the tools are

chmod	Standard tool for specifying protections on objects
chown	Standard tool for modifying the owner of objects
chown	Standard tool for modifying the group of objects
lsacl	Extended tool for listing protections of objects
chacl	Extended tool for changing protections of an object
cpacl	Extended tool for copying protections from one object to one or more objects

ls –l	Used to display the access rights to listed objects

Additional New Protection Capabilities

Domain/OS now provides additional features that enable system administrators and users to limit access to node resources. The features include the "local access only" attribute, the **lprotect**, and node owners' controls.

Local Access Only (LAO)

The **local-access-only** attribute may be used to deny all remote requests to an object. If this attribute is specified in an object's ACL, only local processes will be granted access to the object. This feature is supported only between SR10 nodes; SR9.7 will not respect the LAO attribute of an SR10 object. The **edacl** or **chacl** commands may be used to set or reset this attribute.

The lprotect Command

When a process runs as **root**, the system automatically grants that process full rights to any object, regardless of the access rights that have been assigned to the object. In a network of distributed nodes, the owner of one node may be suspicious of processes on other nodes that claim to be **root**. The **lprotect** command is used to specify whether remote processes running as **root** should be granted this special treatment, and if so, the extent of the special treatment they should be granted. Specifically, **lprotect** can specify that requests from remote **root** processes be treated as follows:

- Do not grant any special rights to a remote requester that is **root**; treat it as if it were "other" or **user.none.none**.
- Grant read rights to a remote requester that is **root**, in addition to any rights applicable to "other" or **user.none.none**.
- Grant all rights to a remote **root** requester.

The following additional restrictions are enforced when **lprotect** is used to limit **root** rights:

- The server process manager (SPM) will not allow processes to be created as **root** unless an active login is done, whereby the user specifies the proper password for the **root** account (that is, **crp –me** as **root** will not be honored).

- Invocations of remote setuid-to-root programs are not allowed.

- Remote requests to setuid-to-root on a local object are disallowed.

You should note that operations on a node configured with links to remote setuid-to-root programs cannot work in this mode. That is, if you use **lprotect** on your node, and you have a link from your node to some setuid-to-root program on another node, you cannot use that program, even though you have an entry for it in a local directory.

The **lprotect** command only applies to the node on which it is issued. It may only be issued by root or by users who have been designated as node owners (see below).

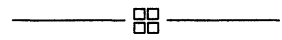
Node Owners

It is often desirable for one or more users to have special privileges on a given node, so that certain operations may only be performed by root or by users who have been designated as **node owners**. The system interprets the ACL on the file **`node_data/node_owners** as specifying which users are to be considered as "owners" of the node; all users with **p** rights to this file are treated as node owners. To prevent subversion of this mechanism, the file **`node_data/node_owners** must itself be owned by root.

Currently, the operations controlled by **`node_data/node_owners** are the following:

lprotect The ability to deny protection overrides to requests by remote roots

sigp The ability to signal any process running on this node, rather than just processes with the same SID



Index

B

C

D

E

F

M

N

O

P

Q

R

S

Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Making the Transition to SR10 Operating System Releases*
Order No.011435-A02

Your Name _____ Date _____

Organization _____

Street Address _____

City _____ State _____ Zip _____

Telephone number (_____) _____

When you use the Apollo system, what **job(s)** do you perform?

- ☐ Programming
- ☐ Application End User
- ☐ Hardware Engineering
- ☐ System Administration
- ☐ Other (describe)_____

How many years of **experience** do you have in using the Apollo system:

_____

What programming **languages** do you use with the Apollo system?

_____

How would you **evaluate** this book?

	Excellent		Average		Poor
Completeness	1	2	3	4	5
Accuracy	1	2	3	4	5
Usability	1	2	3	4	5

Additional Comments: _____

No postage necessary if mailed in the U.S.

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 78 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

fold

Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Making the Transition to SR10 Operating System Releases*
Order No.011435-A02

Your Name _____ Date _____

Organization _____

Street Address _____

City _____ State _____ Zip _____

Telephone number (_____) _____

When you use the Apollo system, what **job(s)** do you perform?

- ☐ Programming
- ☐ Application End User
- ☐ Hardware Engineering
- ☐ System Administration
- ☐ Other (describe)_____

How many years of **experience** do you have in using the Apollo system:

_____

What programming **languages** do you use with the Apollo system?

_____

How would you **evaluate** this book?

	Excellent		Average		Poor
Completeness	1	2	3	4	5
Accuracy	1	2	3	4	5
Usability	1	2	3	4	5

Additional Comments: _____

No postage necessary if mailed in the U.S.

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 78 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

fold

apollo

Making the Transition to SR10 Operating System Releases

011435-A02

OS
OPERATING SYSTEM