* * * DRAFT * * *

POSIX/XL Security

External Specifications

Commercial Systems Division

Location Code:

Project Number:

Print Date: February 20, 1992

Document Control Information
Second Draft : November 23, 1991

*** HP Confidential ***

Copyright © 1992 HEWLETT-PACKARD COMPANY

MPE XL Computer Systems

POSIX/XL Security

External Specifications

HEWLETT PACKARD
19447 PRUNERIDGE AVENUE, CUPERTINO, CA 95014

Part No. POSIX/XL
E1191

Printed in USA 11/91
MPE XL

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company.

Copyright © 1991 by HEWLETT-PACKARD COMPANY

Section 1
POSIX/XL SECURITY

POSIX/XL SECURITY

SECTION 1

1.1 INTRODUCTION

This ES specifies FOS file system access control enhancements being implemented to support POSIX.1 features on MPE XL.

Much of the system behavior described in this ES will be implemented by engineers working in areas other than "security". MPE XL CI commands, MPE XL intrinsics, POSIX.1 C library functions, and POSIX.2 shell commands will be the external interfaces for these new access control features. These external interfaces provide different views of Access Control Definitions (ACDs). With the advent of POSIX.1 features on MPE XL, ACDs are being emphasized as the strategic MPE XL file system access control mechanism. ACDs were introduced in MPE XL release 3.0.

Some of the new security features will only have POSIX.1 C library externals in the first release. These features may be useful to other projects even though they will lack MPE XL CI command or intrinsic externals in the first release. For example, the MPE XL C2 security project may find file mode creation masks a convenient means of implementing maximum (default) file protection.

1.1.1 Problem Statement

Implementing POSIX.1 on MPE XL requires a security mechanism which conforms to the POSIX 1003.1-1988 standard. Existing MPE XL security mechanisms do not satisfy these requirements. POSIX.1 security must coexist with current MPE XL security mechanisms. Users should not be able to use one security mechanism to defeat other security mechanisms. The POSIX.1 security mechanism must interact with MPE XL's file access matrix and ACD mechanisms in a well-defined and useful manner.

1.1.2 Related Specifications

A high-level overview of file system access control enhancements is presented in the POSIX/XL architecture paper. The architecture paper's description may be more appropriate than this ES for business partners and customers who are interested in these security enhancements, but who do not wish to read detailed specifications.

This ES and related external specifications provide detailed information on these security enhancements. The "System Management & POSIX" ES discusses changes to the ALTUSER, LISTACCT, LISTF, LISTFILE, LISTGROUP, LISTUSER, NEWUSER, NEWACCT, PURGE, PURGEACCT, PURGEGROUP, PURGEUSER, and RENAME commands. Separate external specifications have been written to describe the new CHDIR, DISKUSE, NEWDIR, and PURGEDIR commands. The "POSIX.1/XL C Library Functions" ES describes the C library access control interfaces. The "POSIX/XL UID/GID Data Base" design document describes how to create and maintain the user ID and group ID databases. The "Store/Restore for POSIX" ES describes archiving and transporting files. The "POSIX/XL Security" Module Interfaces Document specifies the internal OS access control interfaces implementing the external behaviors detailed in this ES.

1.1.3 Security Goals

The primary technical goal for the first phase of FOS security enhancements is to provide appropriate external security features to support both POSIX. 1 and MPE views of security on the same system. Phase one also attempts to minimize the effort required to implement future extensions by defining appropriate externals. The first release will not attempt to introduce open systems APIs for security, improve ACD performance, or support supplemental GIDs. The proposed changes in ACD behavior are consistent with the POSIX 1003. 6 draft standard and have sufficient flexibility to support extensions to potential de facto standards such as DECorum ACLs.

The file system access control externals proposed in this ES for the MPE XL release supporting phase I POSIX. 1 features successfully address all of the goals listed below:

- Implement POSIX 1003. 1 file and directory access control in an extensible manner consistent with the long-term direction for MPE XL security and industry standards.
- Avoid introducing back doors in file system access control.
- Maintain compatibility with MPE XL 3. 0 security features.
- Minimize the learning curve for existing MPE XL customers by introducing new security mechanisms and concepts only where necessary.
- Minimize implementation cost by using existing MPE XL security features.
- Promote file sharing between applications by permitting file access from both MPE XL intrinsics and POSIX. 1 interfaces.
- Promote incremental migration from obsolete MPE XL and UNIX security mechanisms to long-term security mechanisms.
- Provide the foundation for future open, industry-standard security interfaces and behavior.

1.1.4 Organization of this Report

This security ES is divided into ten parts:

1) Introduction
2) DAC
3) Background
4) MPE XL Access Control Enhancements
5) POSIX. 1 Access Control Interfaces
6) Access Control Interactions
7) Security Commands and Intrinsics
8) Phase One Restrictions

1.2 DAC

Discretionary Access Control (DAC) is the sole topic of this ES. DAC mechanisms enable users to specify how they want to share access to their files. DAC mechanisms provide an environment in which controlled file sharing can take place. The MPE file access matrix and ACDs are examples of DAC mechanisms. DAC is called "discretionary" because it does not impose any restraints on how users may share access to their files. Unlike DAC, Mandatory Access Control (MAC) enforces restrictions on all forms of data sharing.

DAC is only one aspect of MPE XL security. The more general term "security" involves many areas including identification, authentication, audit, labeling, mandatory access control (MAC), and system integrity. DAC is built upon these other security mechanisms. A useful DAC mechanism requires authenticated user identities. Reliable authentication requires password management. Without system integrity other security mechanisms are meaningless.

1.2.1 Generic Model

This subsection presents a very simple, generic DAC model. This generic DAC model is that access control arbitrates access between an active entity called a subject and a passive entity referred to as an object. Successful subject access to an object generally results in a flow of data between the object and the subject. Figure 1-2 presents this generic DAC model:

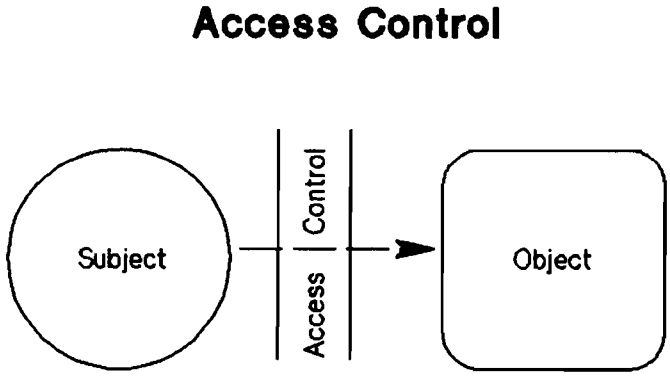

Figure 1-2

This generic DAC model will be used throughout this report to compare MPE XL and POSIX. 1 DAC mechanisms.

1.3 BACKGROUND

This section provides background information necessary for understanding the following section's discussion of DAC enhancements. Released MPE XL DAC features and relevant portions of the POSIX. 1 specification are summarized in this section.

1.3.1 MPE XL Access Control

ACDs and the file access matrix are the primary MPE XL DAC mechanisms. File lockwords are another DAC mechanism, but are less effective because they cannot be audited. (The operating system cannot track who knows a given file lockword. It can only track who has actually attempted to access a file.)

Processes are the only MPE XL subjects. Processes belong to a job or session process tree. All processes in a job or session share a single user identity, capabilities, home group, and logon group. MPE user IDs consist of a user name and an account name. User IDs belong to an account and may only log on to that account. A logon group must be specified on the HELLO command line unless the user has been assigned a default home group. After a user ID has been successfully authenticated by supplying the correct logon passwords, an initial CI process is created for the job or session and assigned the authenticated user ID. All processes in a job or session process tree inherit their identity from the initial CI process.

A process with SM capability is a system manager. System managers have all access to all objects on a system. A process with AM capability is an account manager for its account. An account manager has all access to files in its account.

Files, accounts, MPE groups, and devices are MPE XL file system objects. Only file and device access is controlled by conventional DAC mechanisms. Only ACDs can be used to govern access to devices. File access matrixes cannot be assigned to devices. File access control is described in the following subsections. In addition to being granted access by the applicable DAC mechanism, SF capability is required to create files in the permanent file domain. SM capability is required to create accounts. SM capability or AM capability for an account is required to create users or MPE groups in the account. MPE group save access governs the ability to create files in an MPE group.

When a file is created the user name associated with the job or session to which the process creating the file belongs is saved as the file's creator. Storing an unqualified creator name is one reason files can only be created within the logon account. When files are created they are assigned a default file access mask which grants all access to the ANY file user type. When files are saved in the permanent file domain the FCLOSE securitycode parameter provides a choice between granting all access to the ANY or the CR file user type. HPFOPEN permits an ACD to be assigned to a file when it is created. HPACDPUT assigns an ACD to an existing file. The ALTSEC command manipulates both file access masks and ACDs.

The MPE XL operating system and privileged applications use file privilege levels, negative file codes, and file write-protection to insure system integrity. Although these are DAC mechanisms, they are not available for normal customer use or to unprivileged applications.

Figure 1-3 provides a pictorial summary of unprivileged MPE XL DAC mechanisms.

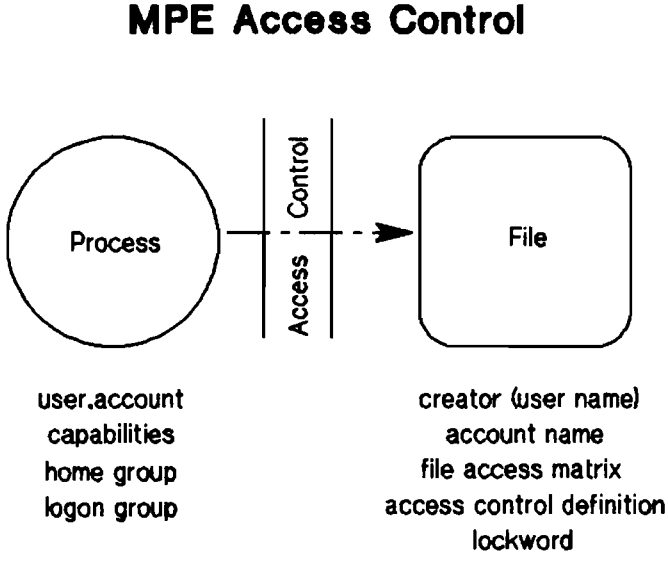

Figure 1-3

1.3.1.1 File Access Matrix

The file access matrix is enforced only when a file lacks an ACD. The access mask assigned to a file's parent account, parent group, and the file itself form the three rows of the file access matrix. Each mask grants file access permissions to any of the file user types which are defined for that access mask level. MPE XL defines six file user types: any user on the system (ANY), account users (AC), account librarians (AL), group users (GU), group librarians (GL), and the file creator (CR). All six file user types are defined for the file-level access mask. All but the CR file user type are defined for group-level access masks. Only the ANY and AC file user types are defined for account access masks. The file access permissions are read (R), lock (L), write (W), append (A), and execute (X). Save (S) access permission may be granted in the group-level access mask to permit file creation in MPE groups. File user types form a hierarchy. Access permissions granted to a file user type are inherited down the hierarchy to the more specific file user types included in the more general file user type. The ANY file user type includes all other file user types. The AC file user type includes all file user types except the ANY file user type. The GU file user type for an MPE group includes all GL file user types defined for that MPE group.

A subject's access to a file protected by the file access matrix is the union of the access granted to all file user types for which the subject qualifies. The account, group, and file masks are ANDed together to determine the access granted to each file user type for a file. Only system managers can assign account access masks. Only system managers and an account's account managers can assign group access masks. Only a file's creator can assign file masks. These three levels of access masks represent a convenient division of responsibility for specifying file access.

Although normal users can't change the account or group access masks which restrict file access, these portions of the file access matrix are not mandatory access controls in that use of the file access matrix itself is discretionary. The RELEASE command can be used to disable the file access matrix. Disabling the file access matrix makes a file accessible to all users on a system regardless of what access is granted by the file access matrix. The SECURE command enables the file access matrix after it has been disabled by the RELEASE command.

File lockwords can be assigned to files to further restrict file access when access is governed by the file access matrix. Only users who can provide the lockword are granted access to the file. Only a file's creator can assign or remove a file lockword. A released file continues to be protected by its lockword. ACDs override both lockwords and the file access matrix.

1.3.1.2 ACDs

The term "ACD" is a proprietary HP term for a concept generally known as an Access Control List (ACL). "ACD pair" is another HP proprietary term for an ACL entry. ACDs have been available on MPE V since V-Delta-4 MIT and more recently on MPE XL since the MPE XL 3.0 release. This report will continue to use HP proprietary terminology since it is describing the MPE XL view of the system.

An Access Control Definition (ACD) is an ordered list of (access mode, user specification) pairs that controls access to an object protected by the ACD. Each entry in the list specifies object access permissions granted to a specific user or group of users. In addition to being granted access to the object protected by an ACD, users may also be granted access to read the ACD itself.

ACDs can be used to control access to files and devices. ACDs override the file access matrix and file lockwords. ACDs permit greater flexibility than the file access matrix in specifying file access because access can be granted to specific users or accounts rather than just to file user types. Users outside the logon account are not all lumped into a single ANY file user type. Each individual user or account may be granted specific access permissions.

The syntax of an MPE XL 3.0 ACD is:

```
ACD                 = (pair_spec)

pair_spec           =  pair_spec                          |
                       pair_spec ; pair_spec              |
                       modes: user_specifications

user_specifications =  user                               |
                       user_specifications set_operator user_specifications

user                =  user_name.account_name             |
                       @.account_name                     |
                       @.@

set_operator        =  , (set union)

modes               =  modes , modes                      |
                       file_access                        |
                       permission

file_access         =  R | W | A | L | X | None

permission          =  RACD
```

The access permissions specified in *modes* are:

- *R* : Read file access
- *W* : Write file access
- *L* : Lock file access
- *A* : Append file access
- *X* : Execute file access
- *NONE* : no access (access denied)
- *RACD* : Read ACD access

User specifications can describe individual users or groups of users:

- *username.accountname* specifies a single user.
- *username.accountname*, *username.accountname*, *username.accountname* specifies a list of three specific users.
- @.*accountname* specifies all users associated with the *accountname* account.
- @.@ specifies all users on the system.

ACDs pairs containing the most specific user specifications are evaluated before ACD pairs containing less specific user specifications. The first pair matched determines a user's access to the object protected by an ACD. A user is denied access if no ACD pair is matched. The order of user specifications from greatest specificity to least specificity is:

- *user_name.account_name*
- @.*account_name*
- @.@

ACDs may contain a maximum of forty pairs.

1.3.2 POSIX.1 Access Control Interfaces

The POSIX. 1 access control model is like other models in that it defines a set of access rules governing the access a process has to a file system object. Every process has a user identity (UID), one or more group identities (GID), a file mode creation mask, and possibly implementation-defined privileges. Every file has a file owner (UID), file group (GID), and a set of file permission bits. The file permission bits grant file access to three process classes: the file owner, file group, and file other classes. A process is in the file owner class for a file if the process' effective UID matches the file's UID. A process is in the file group class if it is not in the file owner class and its effective GID matches the file's GID. A process is in the file other class if it is not in the file owner or file group class. Figure 1-4 illustrates the POSIX. 1 DAC mechanism:

POSIX.1 Access Control

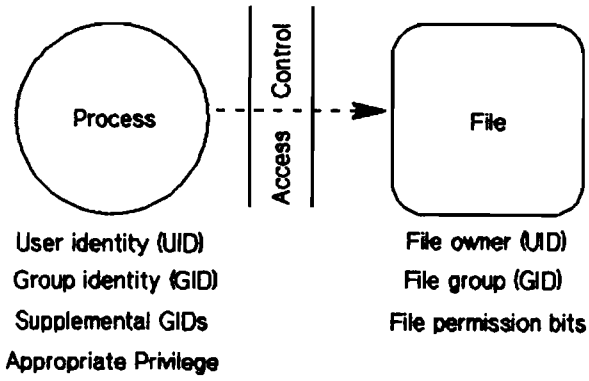

Figure 1-4

POSIX. 1 access control allows read, write, and execute access permissions to be granted to the file owner, file group, and file other classes. This makes a total of nine access permissions which may be expressed for a file or directory in the file permission bits. The POSIX. 1 <sys/stat. h> header file defines a mnemonic bit mask for each of these nine access permissions as well as bit masks for each of the three process classes. This header file also defines a mode__t data type for the file permission bits. Arguments of this data type are used to express access control permissions in the following POSIX. 1 functions:

- initial file permission bits for newly created file system objects are specified by a mode argument passed to open(), creat(), mkdir(), and mkfifo()
- file permission bits are changed by specifying new file permission bits in the mode argument passed to chmod()
- file permission bits are returned by the stat() and fstat() functions in the st__mode member of the stat structure
- a process' file mode creation mask is changed by specifying new file permission bits in the cmask argument passed to umask()

The file permission bits assigned when creating a new file system object are calculated by ANDing the mode argument passed to open(), creat(), mkdir(), or mkfifo() with the one's complement of the file mode creation mask. (The file mode creation mask represents which permissions should *not* be granted.)

The file permission bits are evaluated during access attempts to determine whether access should be granted. The POSIX. 1 functions governed by each access permission are:

- Execute access is checked during directory entry traversal which occurs during pathname resolution of the path arguments passed to the chdir(), chown(), the exec() family, link(), mkfifo(), open(), opendir(), pathconf(), rename(), rmdir(), and unlink() functions. Execute access to a directory is also called search access. Search access to the directories containing each name component in a pathname is required to successfully resolve the pathname.
- Execute access is checked when an exec() family function is invoked to execute a program file.
- Read access is checked when the opendir() or open() function is used to open a directory for reading.
- Read access is checked when the open() function is used to open a file for reading.
- Write access is checked when the open() function is used to open a file for writing or when utime() is used to set file time stamps.

- Write access is checked before a directory entry is created or deleted. The creat(), link(), mkdir(), mkfifo(), open(), and rename() functions create directory entries. Rename(), rmdir(), and unlink() delete directory entries.

POSIX. 1 permits users with appropriate privilege to change their user identity by changing the effective UID and GID of a process via the setuid() and setgid() functions. A process' effective UID or GID may also be changed by an exec() of a program with the set-user-ID or set-group-ID bits enabled. File and directory UIDs and GIDs can also be changed via the chown() function.

Processes with appropriate privilege are granted requested access to an object without evaluating the object's file permission bits. File execute access is the single exception to this statement. Because POSIX doesn't provide a means of distinguishing regular files containing executable scripts from regular files containing other types of data, POSIX. 1 states that the file permission bits should be checked to verify execute access has been granted to one of the file classes before granting execute access to a process with appropriate privilege. Execute access being granted to one of the file classes is used as an indication that the file contains executable statements.

The access() function can be used to check file or directory accessibility using the real UID and GID rather than the effective UID and GID. Since the real UID may be different than the effective UID, actual access may be different than indicated by the access() function.

1.4 MPE XL ACCESS CONTROL ENHANCEMENTS

File system access control must be enhanced to accommodate new POSIX-compatible MPE XL FOS features. Access to files created outside MPE groups and to hierarchical directories must be controlled by the operating system. A subset of FOS access control features must support POSIX.1-compatible access control.

Relatively few new FOS access control features are being introduced to provide POSIX.1-compatible access control. The most significant changes are in generalizing how existing MPE XL features are conceptualized. These new concepts provide a more general view of existing MPE XL access control behavior and permit small enhancements to be added as natural extensions to the new concepts. Understanding how these new concepts explain current MPE XL behavior is the principal learning curve. Since the new generalized concepts can be related to familiar system behavior, the learning curve should not be very steep. Once the new concepts are understood, understanding the new extensions should only require a small incremental effort.

System administrators will only have to learn these new generalized concepts and the few natural extensions. They will not have to be familiar with POSIX.1 access control concepts. Only C programmers writing code which calls C library functions will have to learn POSIX.1 access control concepts. This requirement is common to all C programming environments in that these access control concepts are inherent in the definition of the C library functions.

File system access control can be summarized very briefly. During system boot, while the system is not yet up, all access will be granted to all file system objects. Once the system is up, file privilege level checking and priv-mode (negative file code) file protection will continue to be checked before other access control mechanisms. SM and AM capability will be checked before ACDs or the file access matrix. ACDs will continue to override file lockwords and the file access matrix. File privilege, negative file codes, write protection, and lockwords will be enforced only for files in MPE groups for first release. A process will be required to provide lockwords for files protected by active lockwords whether or not the process has appropriate privilege. Write protection (flab^.file_flags.protected) will deny write access to objects. SF capability will be required to create entries in directories in the permanent file domain. Most operations currently restricted to file creators will be available to users with appropriate privilege or sufficient access. There will no longer be a blanket prohibition on creating files outside the logon account.

1.4.1 Object Ownership

The basis for tracking file and directory ownership needs to be enhanced. MPE XL has used the creator name, an unqualified user name, to track file ownership. The creator name for the root directory, MPE groups, and accounts was not recorded. Only files were assigned creator names. Since files and hierarchical directories can now be created outside the logon account, unqualified user names are no longer sufficient for indicating object ownership. Object ownership must be indicated by a user identity unique to at least the local system.

For first release file ownership for all newly created or renamed files will be indicated by a fully qualified user name. This fully qualified name will be referred to as the file owner or file user ID (UID). The file owner is a generalization of the file creator. The file creator was a static value for the lifetime of a file. The file owner can be changed during the lifetime of a file by calling the new HPFSETOWNER intrinsic. Several rules restrict file ownership changes. For further details see the intrinsics reference section of this ES.

File owners will be assigned to all newly created files and directories. The file owner of the root directory after an install will be MANAGER.SYS. Directories created before installation of the new FOS release

will lack file owners since older MPE XL releases did not initialize ownership information. Directories with uninitialized file ownership information will appear to have a file owner of "0" when displayed by LISTFILE. The UID database interfaces will reserve zero UID values for use by MPE XL. Zero UID values will not be permitted to be assigned to users, files, or directories. File owners may be assigned to old directories by calling the HPFSETOWNER intrinsic.

Object ownership for MPE groups, accounts, and the root directory are new concepts. The existing access control policy for these directory types is based solely upon appropriate privilege. Account managers did not retain any additional access to MPE groups they had created if their AM capability was removed by their system manager. Starting with the FOS release introducing POSIX. 1 features, the ability to create or delete entries in the root directory, MPE groups, and accounts will no longer be based solely on appropriate privilege. Directory file owners will be granted all access to the directories they own.

1.4.1.1 Backwards Compatibility

As long as file ownership is not changed by calling HPFSETOWNER, the file owner can be thought of as being a fully qualified file creator name which is not restricted to its logon account.

1.4.2 Sharing Objects

Files created under the root directory or below some combination of hierarchical directories below the root directory will not be within an MPE account. Since MPE accounts provided the basis for file sharing prior to the introduction of POSIX. 1 features, MPE XL's file sharing concepts need to be generalized. (All file user types other than the ANY file user type were members of the logon account.)

File sharing will be generalized using the POSIX. 1 file group ID (GID) concept. When files and directories are created they will be assigned their parent directory's file group ID (GID). MPE accounts will be assigned a unique GID when they are created. The HPGID database will record this association of MPE account and file GID. The HPFSETOWNER intrinsic can be used to change an object's GID.

For first release HPFSETOWNER will not change MPE group or account GIDs. Since these GID values will not change after an account is created, access control to MPE groups and accounts will not change from previous releases.

Root directories created before installation of the new FOS release will lack file GIDs since older MPE XL releases did not initialize file GID information for the root directory. Uninitialized file group information will appear as a file GID of "0" when displayed by LISTFILE. The GID database interfaces will reserve zero GID values for use by MPE XL. Users, files, and directories may not be assigned zero GID values. File group values may be assigned to old directories by calling the HPFSETOWNER intrinsic.

1.4.2.1 Backwards Compatibility

MPE XL file sharing will operate exactly as it has in the past if new POSIX. 1-related features are not used (no files or directories are created outside MPE accounts and the HPFSETOWNER intrinsic is not used to change file GIDs).

1.4.3 Appropriate Privilege

SM capability will permit a process unrestricted access to all file system objects. AM capability will also permit unrestricted access to a file system object if the object's GID matches the process' GID. An account manager's appropriate privilege is no longer restricted to the manager's logon account. Nor is an

account manager guaranteed appropriate privilege for all objects in his/her account. See the following "Creating Objects" subsection for a further discussion of GID-based appropriate privilege.

File write protection, negative file codes, file privilege, and lockwords may prohibit access to files in MPE groups even for processes with appropriate privilege. Lockwords must be provided for files protected by active lockwords regardless of whether a process has appropriate privilege. Lockwords don't really provide any access control for processes with appropriate privilege since processes with appropriate privilege can read a file's lockword and then provide the lockword to satisfy access control rules.

File execute access will not always be granted to processes with appropriate privilege. The POSIX.2 shell relies upon execute access being granted only to shell scripts and program files. To accommodate the POSIX.2 shell, execute access will be granted to processes with appropriate privilege when:

- The file being accessed has an NMPROG, NMXL, PROG, or SL file code.
- The file has an ACD granting execute access to one or more users.
- The file is protected by a file access matrix granting one or more file user types execute access.

Granting execute access only in these circumstances is backwards compatible with existing MPE XL behavior. MPE XL only checks execute access when job files are streamed or program files are loaded. Processes with appropriate privilege will be granted execute access to program files due to their file codes. Processes with appropriate privilege will be able to stream job files because they will be granted read access to job files. Read or execute access enables a process to stream job files.

1.4.4 Directory Access

Prior to this release MPE XL controlled access to directories largely through the AM and SM capabilities. File creation was the single exception controlled by save access expressed in MPE group access masks. File creation outside the logon account was prohibited by MPE XL CI commands and intrinsics. All users could read all file names on a system via LISTF or LISTFILE. File-level access control in the form of the file access matrix or ACDs granted or denied file access.

Starting with this release access to directories will be controlled by directory ACDs. See the following "ACD Changes" subsection for the syntax of directory ACDs. The ability to grant or deny directory access, rather than just file access, will make expressing access control more convenient. For first release directory ACDs will govern access only to hierarchical directories. Directory ACDs will not be supported for MPE groups, accounts, or the root directory.

Four directory access permissions may be granted to users:

- Permission to **read directory entries** is required to read the names contained in a directory. Read directory entries access is only required when names must be read from a directory. Access to a specified name does not require read directory entries access.
- Permission to **traverse directory entries** is required to use a name in a directory entry to refer to a file system object or its attributes. File name resolution requires traverse directory entries access. Accessing any file attribute other than a file's name also requires traverse directory entries access for the directory containing the directory entry.
- Permission to **create directory entries** is required to insert entries into a directory. Creating files or directories in the permanent file domain involves creating a directory entry in the parent directory. Renaming files also involves creating a directory entry.
- Permission to **delete directory entries** is required to delete entries from a directory. Removing files from the permanent file domain involves deleting a directory entry. Renaming files also involves deleting a directory entry.

Directory ACDs will permit these directory access permissions to be directly expressed for hierarchical directories. Access to MPE groups, accounts, and the root directory will be controlled by a generalization of the existing access control rules. These rules can be stated using the new directory access permissions as:

- All users will be granted read and traverse directory entries access to MPE groups, accounts, and the root directory.
- Appropriate privilege will be required to create or delete directory entries from an account or the root directory.
- Create directory entries access for MPE groups will be governed by the save access permission expressed in the group access mask. Account-level access masks will implicitly restrict save access to the AC file user type since save access can't be expressed in these masks. Only processes with SM capability will be able to create files in an MPE group in an account different than the logon account. (Users lacking SM capability may be able to create files/directories in hierarchical directories outside their logon account subject to the previously described access control rules for hierarchical directories.)
- Delete directory entries access for an MPE group will map to one of two different requirements depending on whether the directory entry being deleted references a file or a hierarchical directory. Removing a directory entry referencing a hierarchical directory from an MPE group will require save access to the group. Removing a directory entry referencing a file from an MPE group will require write access to the file. File write protection, negative file codes, file privilege, and lockwords will be checked before permitting a file's directory entry to be removed from an MPE group.

1.4.5 File Access Matrix

The MPE XL file access matrix will continue to behave as it has prior to the introduction of POSIX. 1 features. The file access matrix is not being generalized to the hierarchical directory because an N-level file access mask would be difficult to understand and unwieldy to use. Files protected by the file access matrix must reside in MPE groups since the file access matrix continues to be defined in terms of exactly three access masks. ACDs will continue to override the file access matrix.

A process' current working directory will not affect the file user types in which the process is a member. A process will not be a member of the GU file user type for unqualified file references (eg. "A") when the CWD references a directory other than the home or logon group. Membership in the GU file user type will continue to be defined by the job's or session's home and logon groups. Logging on and its variants such as :CHGROUP and AIFCHANGELOGON will remain the only way to establish a logon group. Group passwords will continue to be the authorization mechanism for establishing a logon group. Since the GU user type is not defined relative to the CWD, a process' CWD may be set to any MPE group without providing group passwords. Traverse directory entries access to all directories specified in the pathname passed to chdir() will be the only access required to set a process' CWD.

The file access matrix only applies to files whose GID matches their parent account's GID. If the GID is changed to a different value or if the file is moved to a different account an ACD will be automatically assigned. (See the "ACD Changes" subsection for further information on automatic ACD assignment.) Conceptual consistency between the file access matrix and ACDs can be maintained by thinking of the AC file user type as being file GID-based.

The CR file user type has been generalized to refer to the file owner rather than the file creator. The file owner does not have to be a member of the account associated with the file's GID.

Files will be assigned a file access mask regardless of where they are created. (Another aspect of the "files are files" goal.) Files created outside MPE groups will be assigned the file access mask "R,W,L,X,A:ANY". Files will also be created with the file label flags_t.released boolean set to false (secured). For all practical purposes hierarchical directories will not have file access masks. MPE XL should not provide a means of viewing or manipulating a hierarchical directory's file access mask. The only exceptions to this statement should be raw dumps of the file label provided by provided by ":LISTFILE ,-1" and debuggers.

The :RELEASE and :SECURE commands may be used to manipulate the released bit in any file's file label. The state of the released bit will have no effect until the file is linked to an MPE group and lacks an ACD. Since the file access matrix doesn't protect directories, the :SECURE and :RELEASE commands will not operate on directories. Directories will not appear to have file access masks and will be invalid targets for operations which manipulate the file mask.

1.4.6 File Lockwords

Lockwords may only be assigned to files. The FRENAME, FOPEN, and HPFOPEN intrinsics will return errors when attempts are made to assign lockwords to directories. The security advisor routines will only enforce a file lockword when the file is in an MPE group and lacks an ACD.

All users will be required to provide lockwords for files protected by active lockwords. Appropriate privilege by itself will be insufficient to access files protected by active lockwords. Processes with appropriate privilege will be required to provide correct lockwords before accessing lockword-protected files. This decision has been made to remain compatible with present file lockword behavior.

Pathname syntax does not support file lockwords. Lockwords must be supplied by either being embedded in MPE syntax file names or in response to lockword prompting.

1.4.7 ACD Changes

ACD assignment may be affected by interactions with the process file mode creation mask. See the "Access Control Interactions" section for further information.

MPE XL ACDs will be enhanced in six areas:

1) Objects which cannot be protected by the file access matrix will be required to have ACDs. Only files whose file GID matches their parent MPE group's GID may be protected by the file access matrix. Only optional ACDs may be deleted (or have all of their ACD pairs deleted). Required ACDs must contain at least one ACD pair.
2) ACDs will be automatically assigned to objects lacking ACDs when an object ceases to be a candidate for file access matrix protection.
3) ACDs may be assigned to hierarchical directories. Directory ACDs will grant directory access permissions.
4) New user specifications in ACD entries will enable ACDs to act as an additional POSIX.1 access control mechanism.
5) File ACDs containing new user specifications and directory ACDs will be assigned a new ACD version number.
6) Pathnames may be used in specifying ACD operations.

1.4.7.1 Automatic Assignment

Automatically assigned ACDs will specify (RACD:@.@). File owners and processes with appropriate privilege will be able to access the object. All other processes will only be able to read the ACD. File system operations which will cause automatic ACD assignment are:

- Creating a directory. Directories are never protected by the file access matrix. Only hierarchical directories will be assigned ACDs in first release. MPE groups, accounts, and the root directory will be protected by the previously described access control rules.
- Creating a file in the permanent file domain outside an MPE group without specifying an ACD as part of the creation operation. HPFOPEN is currently the only intrinsic which permits an ACD parameter to be specified when creating an object. (The file access matrix only protects files in MPE groups.)
- Renaming a file from an MPE group to a directory which is not an MPE group. (The file access matrix only protects files in MPE groups.)
- Renaming a file from an MPE group to an MPE group outside the original account. (The file access matrix only protects files in MPE groups when the file's GID matches the parent group's GID.)
- Changing the GID of a file protected by the file access matrix. (The file access matrix only protects files in MPE groups when the file's GID matches the parent group's GID.)

1.4.7.2 Directory ACDs

Directory ACDs will allow four directory access permissions in addition to RACD to be granted:

- RD - **read directory entries**
- TD - **traverse directory entries**
- CD - **create directory entries**
- DD - **delete directory entries**

1.4.7.3 New User Specifications

ACDs will support new $OWNER and $GROUP user specifications to refer to the an object's current file owner (UID) and file group (GID). The $OWNER user specification will enable file owners to voluntarily limit their access to an object. For example, file owners may grant themselves read-only access to a file to guard against accidentally modifying the file. The $GROUP user specification permits a dynamic reference to the object's file GID. This dynamic reference makes it unnecessary to modify an ACD to correct file sharing via an @.account ACD pair when an object's GID is changed. A file group mask will also be supported as a new $GROUP_MASK user specification to enable ACDs to act as an additional POSIX.1 access control mechanism. The $GROUP_MASK is discussed further in the following "ACD Evaluation" subsection and in the "Access Control Interactions" section. The $OWNER, $GROUP, and $GROUP_MASK user specifications are not permitted in device ACDs.

1.4.7.4 ACD Versions

ACDs containing directory access permissions or the new subject identities will have a different version number than MPE XL 3.0 ACDs. File ACDs which do not contain new information will have the same version number as MPE XL 3.0 ACDs. Older MPE XL releases will politely refuse to evaluate or modify unrecognized ACDs, but will be able to delete these ACDs. A file's creator, users with SM capability, and users with AM capability for the file's account will be able to access files protected by unrecognized ACDs since these older MPE XL releases do not evaluate ACDs to determine file access for these users.

1.4.7.5 Pathname Support

The :ALTSEC command will be enhanced to support MPE-escaped semantics for its filereference parameter. The HPACDPUT and HPACDINFO intrinsics will be extended to allow file system objects to be referenced using either MPE-escaped or POSIX semantics. The HPFSETOWNER intrinsic will support all three file name semantics.

1.4.7.6 Enhanced Syntax

The syntax of an enhanced ACD is:

```
ACD                 = (pair_spec)

pair_spec           =  pair_spec                        |
                       pair_spec ; pair_spec            |
                       modes: user_specifications

user_specifications =  user                             |
                       user_specifications set_operator user_specifications

user                =  user_name.account_name           |
                       @.account_name                   |
                       @.@                              |
                       $OWNER                           |
                       $GROUP                           |
                       $GROUP_MASK

set_operator        =  , (set union)

modes               =  modes , modes                    |
                       file_access                      |
                       dir_access                       |
                       permission

file_access         =  R | W | A | L | X | None

dir_access          =  CD | DD | RD | TD | None

permission          =  RACD
```

1.4.7.7 ACD Evaluation

MPE XL 3.0 ACD evaluation matched exactly one ACD pair (3.0 ACD pairs were exclusive). This remains true for enhanced ACDs with one exception. A process may match both a $GROUP and an @.account ACD pair when the account referenced by the @.account ACD pair is associated with the same GID as the file GID. When more than one ACD pair is matched, the union of the access permissions granted by the ACD pairs matched is granted to the process.

ACDs pairs containing the most specific user specifications are evaluated before ACD pairs containing less specific user specifications. A user is denied all access if no ACD pair is matched. The order of user specifications from greatest specificity to least specificity is:

- $OWNER
- *user_name.account_name*
- $GROUP
- *@.account_name*
- @.@

The $GROUP_MASK is not directly matched, it is used to restrict the access permissions granted when one or more ACD pairs belonging to the file group class is matched. All ACD pairs other than $OWNER and @.@ belong to the file group class.

1.4.7.8 Examples

A few ACD examples may help clarify these concepts:

- (RACD,W:JOHN.DOE; W:@.DOE,@.PAYROLL; R:@.@) grants read ACD access to the user JOHN.DOE; write access to all users in the DOE and PAYROLL accounts; and read access to all other users on the system.

- (NONE : JIM.DOE, @.ACCTING ; R,W : @.@) denies all access to the user JIM.DOE and all users in the ACCTING account; and grants read and write access to all other users on the system.

- (CD,DD,TD,RD,RACD:$GROUP; TD,RD,RACD:@.@) grants users in the directory's file group class the ability to create, delete, read, and traverse directory entries as well as read the ACD; and all other users on the system the ability to read and traverse directory entries as well as read the ACD.

- (RACD,R,W,L,A,X:$OWNER; RACD:$GROUP,$GROUP_MASK,@.@) grants all file access to users in the file owner class. File access permissions for the four POSIX ACD pairs (discussed later) are specified in this example ACD.

1.4.8 Creating Objects

MPE XL has historically restricted file creation to MPE groups within the logon account. This restriction will be removed from FOS. Users will be able to create files/directories outside their logon account in any directory to which they can traverse and have been granted create directory entries access. Although the primary reason for removing this restriction is to support POSIX.1-compatible features, the ability to create files outside the logon account has been an enhancement requested by MPE XL customers. When hierarchical directories are not created on a system, only users with SM capability will be able to create files outside their logon account.

Restoring a file recreates an object with its old attributes as long as the file is restored to its old location in the directory. RESTORE;LOCAL creates a new object.

Account managers will need to understand that the appropriate privilege granted by AM capability is GID-based when the new ability to create files across account boundaries is used. A process can call HPFSETOWNER to change the GID of a file created outside its logon account to its effective GID (its logon account's GID). This moves the file from the administrative domain of one account manager to the domain of the process' account manager. Being unable to manage the file is somewhat awkward for the first account manager since the disk space used by the file continues to accrue to the account in which it resides. Account managers may need to cooperate with one another or with their system administrator to manage disk space in an environment where files are being created across account boundaries. This is a natural side effect of granting object creation access across account boundaries (which are the basis for resource accounting).

1.4.9 Renaming Objects

A file's creator will not be the only user able to rename or move a file between directories in the permanent file domain. Users with sufficient access will be able to rename a file as long as the file doesn't have an active lockword. Only a file's creator will be able to change a file's lockword. Rename operations in which either the source or target names includes a lockword will be restricted to the file's creator. Renaming a file requires DD access to delete the old directory entry and CD access to create the new directory entry. TD access is also needed to all name components in the source and target file names. SF capability is also required when the new entry is created in a permanent file directory.

Renaming a file does not create a new object. Renaming a file changes the existing file's directory link. Renaming a file will also convert unqualified creator names to fully qualified file owner names. See the File System ES for details on the FRENAME intrinsic.

1.4.10 Reporting Object Access

Access control information will continue to be reported using the LISTF and LISTFILE commands. LISTFILE and LISTF will display access control information for directories as well as for files. The creator field displayed by the DETAIL;PASS (-3) format will show a fully qualified user.account file owner identity when the label contains this information. An object's GID will also be displayed by the LISTFILE,-3 format. The ACD field will continue to display "NO ACD ACCESS" when a process lacking RACD attempts to display an ACD. (The MPE XL 3.0 :LISTFILE command with format option -2 (ACD) requires AM or SM capability due to a software defect.) All processes will continue to be able to display their access to a file system object using format option 4 (security) regardless of whether they possess RACD access. Processes with RACD access will be able to obtain ACD information using the

HPACDINFO intrinsic. See the File System ES for detailed information on the FLABELINFO and FFILEINFO intrinsics.

1.4.11 Enhanced MPE XL DAC Model

Although this section may have seemed somewhat involved, there are actually few changes to the MPE XL DAC model. It is mostly the terminology and how the MPE XL DAC mechanisms are viewed which has changed. Figure 1-5 illustrates the enhanced MPE XL DAC model:

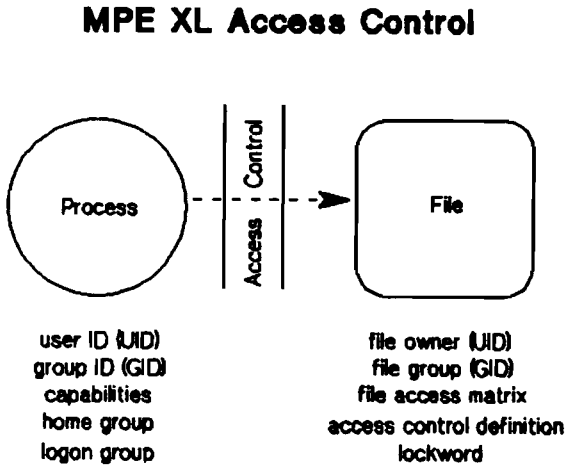

Figure 1-5

Basing file ownership on the file owner ID (UID) and file sharing on the file group ID (GID) are the fundamental changes in the MPE XL DAC model. Removing creator-only restrictions on most file system operations is another basic change. ACD enhancements to support directory access control and the new user specifications round out the DAC changes.

1.5 POSIX.1 ACCESS CONTROL INTERFACES

1.5.1 Design Overview

The fundamental design decision is to implement POSIX. 1 security using MPE XL Access Control Definitions (ACDs). The POSIX. 1 file permission bits will not be implemented as a separate access control mechanism. POSIX/XL C library functions will support the file permission bits as a view of the ACD mechanism. Figure 1-6 illustrates how POSIX. 1 file permission bits will be implemented as a view of the ACD mechanism:

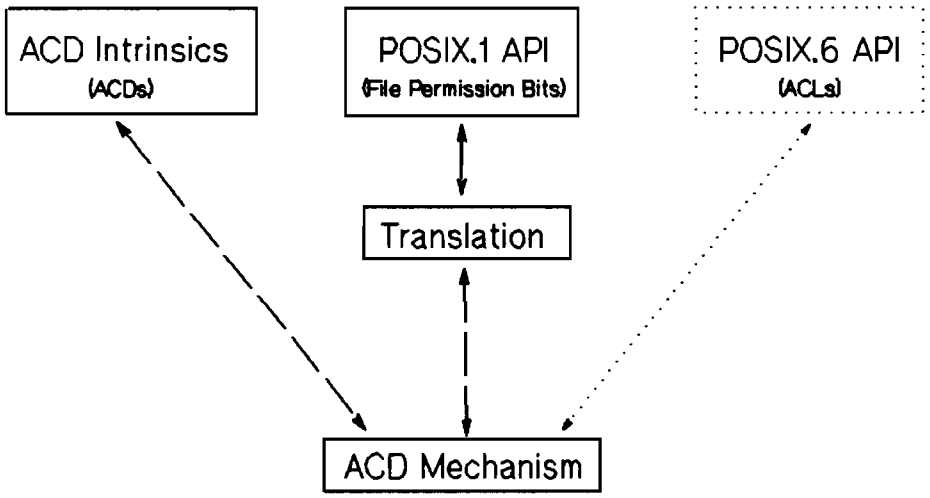

Figure 1-6

When a POSIX. 1 interface such as open() creates a file, an ACD will be assigned to the file as part of the file creation operation. When the POSIX. 1 chmod() function is invoked to set access permissions for a file or directory, ACD information will be manipulated. If the file doesn't have an ACD, chmod() will assign an optional ACD to the file. If an ACD already exists for the file, the ACD will be modified. POSIX. 1 applications and POSIX. 2 commands will continue to use file permission bits to specify access permissions and will be unaware that file permission bits are implemented as an interface to the ACD mechanism. ACDs will be enhanced as described in the previous section to enable the ACD mechanism to operate as a POSIX. 1 additional access control mechanism and to provide directory access control.

Implementing POSIX. 1 access control using ACDs tightly integrates POSIX security with MPE XL security and minimizes:

- risk of introducing security back doors
- additional complexity introduced in the MPE XL security policy
- customer learning curves
- documentation costs
- engineering investment by simplifying the security design

1.5.2 Implementation

All subjects (processes) and objects (files or directories) have a user ID (UID) and a group ID (GID). After a user has been successfully authenticated by supplying the correct logon passwords, an initial CI process is created. The authenticated user name is used as a key to fetch the UID and GID to be assigned to the CI process from the UID and GID databases. All processes in a job or session process tree inherit their UID and GID from the initial CI process; processes will not be able to change their UIDs or GIDs. First release will not support setuid(), setgid(), set-user-ID programs, or set-group-ID programs. All processes in a job/session process tree will have the same UID, GID, and appropriate privileges. There will not be distinct real, effective, or saved-set UIDs or GIDs in phase I. Supplemental GIDs will not be supported in first release. File system access control will use the job/session user.account name as a symbolic replacement for the numeric process UID. The job/session account name will be used as a symbolic replacement for the numeric process GID. Processes will have numeric UIDs and GIDs, but these numeric values will be used only for determining access to processes via the signal functions. One side effect of these phase I restrictions is that a process' GID will always be the same as the GID associated with its logon account.

When an object is created it is assigned the UID of the process creating the object and the GID of the object's parent directory. POSIX.1 allows implementations to inherit GIDs from either the parent directory or the creating process. MPE XL inherits GIDs from the parent directory. GID inheritance from the parent directory makes GID-based file sharing backwards compatible with with account-based file sharing. GID inheritance from the parent directory supports more flexible file sharing and is compatible with SunOS and OSF/1. Multi-platform application developers should be made aware that MPE XL GID inheritance differs from HP-UX which inherits GIDs from the creating process. IBM's AIX permits GID inheritance to be configured either way as a directory attribute.

SM and AM capability will act as appropriate privileges for file system access control. Except for file execute access for which POSIX.1 specifies special rules, processes with SM capability will be granted all access to all files and directories on the system. Processes with AM capability will be granted all access to all files and directories whose GID matches the process' GID. File execute access is granted to processes with appropriate privilege when execute access has been explicitly granted by the DAC mechanism controlling access to the file (either an ACD or the file access matrix) or when the file's file code is is NMPROG, PROG, NMXL, or SL. The executable file codes are an MPE XL extension consistent with the intent of the POSIX.1 specification.

When a file or directory is created using a POSIX/XL C library routine, the file permission bits from the function's mode argument will be masked (restricted) by the process' file creation mode mask (cmask) and then translated into an ACD. This ACD will be passed to HPFOPEN and assigned to the file or directory being created.

Translation of the file permission bits to ACD pairs is very simple. ACDs created by POSIX/XL C library routines will contain the four ACD pairs $OWNER, $GROUP, @.@, and $GROUP_MASK. The file permission class bit masks defined by POSIX.1 for the <sys/stat.h> header file are used in the following description of this mapping. The file owner class (S_IRWXU) bits are mapped to a $OWNER ACD pair, the file group class (S_IRWXG) bits are mapped to a $GROUP_MASK and a $GROUP ACD pair, and the file other class (S_IRWXO) bits are mapped to an @.@ ACD pair. All four ACD pairs are also granted RACD access. Figure 1-7 illustrates the mapping between POSIX.1 file permissions and ACD access permissions:

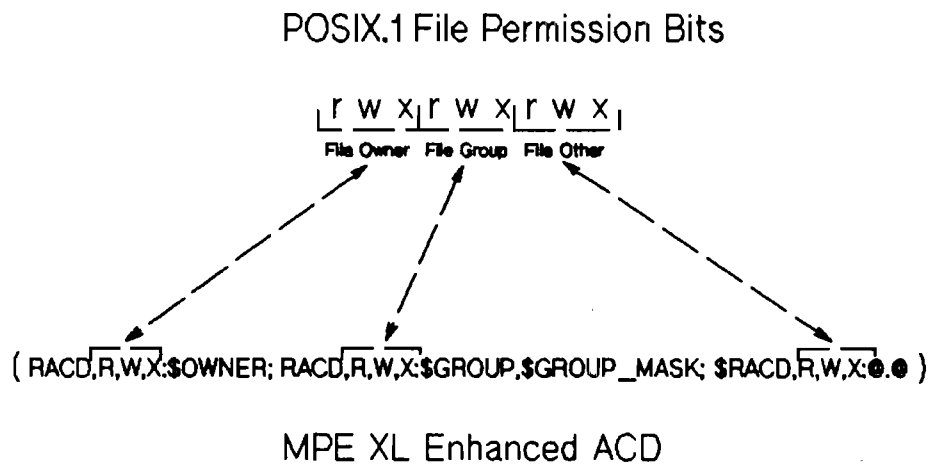

Figure 1-7

Chmod() modifies the access permissions granted in these ACD pairs using this mapping between file permission bits and ACD pairs. Chmod() assigns an ACD when the target file specified in chmod()'s path argument lacks an ACD. If the target object has an ACD, chmod() modifies the four POSIX. 1 ACD pairs. Chmod() always grants RACD to the four POSIX. 1 ACD pairs. Chmod() modifies the $GROUP pair only if the ACD contains exactly the four $OWNER, $GROUP, $GROUP_MASK, and @.@ pairs. This behavior is discussed in more detail in the following "Access Control Interactions" section. File ACDs assigned by chmod() will override the file access matrix and file lockwords. Chmod() will fail if the target object is a is an MPE group, account, or the root directory since the first release will not support ACD assignment to these directory types.

Stat() and fstat() map the access permissions granted by an ACD into file permission bits using this mapping in reverse. Rules governing mapping between the file group class permission bits and the access permissions in the $GROUP_MASK entry are also discussed in the "Access Control Interactions" section.

POSIX. 1 directory read and execute access permissions map directly to the RD and TD directory ACD access permissions. The POSIX. 1 directory write access permission will be mapped to the combination of create directory entries (CD) and delete directory entries (DD) access permissions. When the file permission bits being assigned to a directory grant write access to a file class, both CD and DD will be granted in the directory ACD to the corresponding ACD subject.

The process file mode creation mask manipulated via the umask() function will be stored directly as file permission bits in a process-local data structure.

1.6 ACCESS CONTROL INTERACTIONS

1.6.1 File Lockwords

Lockword prompting will not be performed for any POSIX/XL C library functions. Files with active lockwords will be inaccessible via the POSIX/XL C library interfaces until either the lockword is removed or an ACD is assigned to the file.

1.6.2 Files in MPE Groups

Appropriate privilege will not grant access to files in MPE groups protected by file lockwords, negative file codes, or file privilege. Write protection will deny write and append access to a file and deny DD access to a file's directory entry. Negative file codes, file privilege, and write protection act as additional access control mechanisms as defined by the POSIX.1 specification. File lockwords are not a POSIX.1-compatible feature since they do not coexist with file ACDs (the MPE XL implementation of POSIX.1 access control) and they do not satisfy the requirements of an alternate access control mechanism. Files in MPE groups protected by negative file codes, file privilege, or lockwords will not be accessible via POSIX/XL C library routines since these interfaces do not support lockword, privilege, or file code parameters.

1.6.3 ACD POSIX.1 Compliance

An application calling chmod() expects the access permissions it assigns to a file or directory to be enforced. This is why POSIX.1 permits an additional access control mechanism to only further restrict access defined by the file permission bits. An additional access control mechanism cannot grant access not granted by the file permission bits. The $GROUP_MASK ACD entry was introduced to reconcile more expressive ACDs with the less expressive file permission bits in a POSIX.1-compatible (and POSIX.6-compatible) manner. The $GROUP_MASK ACD entry enables chmod() to modify access without destroying more expressive ACDs.

The access permissions specified in the $GROUP_MASK ACD entry define the most permissive access which will be granted by ACD entries in the file group class. The $GROUP_MASK access permissions restrict the access granted by all ACD entries in the file group class. When evaluating an ACD entry belonging to the file group class, only access permissions granted by both the ACD entry matched and the $GROUP_MASK ACD entry are granted. The $GROUP ACD entry cannot be used to restrict the file group class because the access granted in the $GROUP ACD entry may be less than the access granted in other ACD entries belonging to the file group class. For example, a specific user may be granted read and write access when $GROUP is granted only read access. For first release the file group class consists of all *user.account*, $GROUP, and *@.account* ACD entries. In other words all ACD entries other than $OWNER, @.@, and $GROUP_MASK.

When an application invokes chmod() to set file permission bits for an ACD containing ACD pairs other than the four POSIX ACD entries, the file group class permission bits are used to set the permissions granted by the $GROUP_MASK ACD entry while the $GROUP entry is not changed. This restricts all ACD entries in the file group class to the access specified by the chmod() call without destroying the access granted by ACD entries chmod() knows nothing about. When an application invokes chmod() to set file permission bits for an ACD containing only the four POSIX ACD entries, the file group class permission bits are assigned to both the $GROUP_MASK and $GROUP ACD pairs. This difference in behavior enables granting more permissive access to work in the case where the $GROUP pair is the only member

of the file group class. Using chmod() to grant more permissive access doesn't always work in the case of more expressive ACDs because other ACD entries in the file group class may restrict access. A reasonable mapping of the more permissive file permission bits to the different file group class ACD entries doesn't exist. This problem is inherent in using a less expressive interface to manipulate a more expressive mechanism.

Consider the unfortunately not uncommon sequence of POSIX. 1 calls:

1) Call stat() to get the current file permission bits. (For this example assume read, write, and execute access are granted to the file group class.)
2) Call chmod() with file permission bits set to zero to deny access to the file.
3) Access the file in a pseudo-exclusive manner.
4) Call chmod() to restore the file permission bits read by stat().

The intent of these steps is to obtain pseudo-exclusive access to the file while the file permission bits are set to zero and then to restore file access to its state prior to the first chmod() call. Setting only the $GROUP_MASK ACD entry's access permissions to zero when chmod() is called the first time permits file access to be properly restored by resetting the $GROUP_MASK ACD entry's access permissions when chmod() is called the second time. As long as the $GROUP_MASK ACD entry denies read, write, and execute access, access granted by other ACD entries will be restricted by the $GROUP_MASK. When chmod() is called to restore file access, granting read, write, and execute access in the $GROUP_MASK ACD entry restores the ACD to its original state. In this way the $GROUP_MASK ACD entry provides a window for chmod() into the more expressive file group class ACD entries.

ACDs may be missing any or all of the four POSIX ACD entries since ACDs created using MPE interfaces are not required to contain any particular ACD entries. ACDs missing any of these four ACD entries are still POSIX. 1 compliant since they can be regarded as containing default values for the missing ACD entries. MPE XL ACD *evaluation* is consistent with default values of all access for missing $OWNER and $GROUP_MASK entries and no access for missing $GROUP and @.@ entries. If an ACD contains an @.@ entry and lacks a $GROUP entry, members of the file group class will match the @.@ entry.

1.6.4 SF Capability

A process must have Save Files (SF) capability to create an entry in a directory node (root, account, MPE group, or hierarchical directory) in the permanent file domain. In POSIX. 1 terminology, SF capability will act as an additional access control mechanism. POSIX/XL C library functions will only create file system objects in the permanent file domain. A process must therefore have SF capability to successfully call the POSIX. 1 open(), creat(), mkdir(), mkfifo(), and rename() functions.

1.6.5 Cmask Interactions

A process' file mode creation mask (cmask) affects ACD assignment once the cmask is initialized. Uninitialized process cmasks do not affect ACD assignment. All processes will have a file mode creation mask. This mask will be uninitialized in the initial CI process. A child process inherits its parent's cmask. The POSIX/XL C library umask() function may be called to set a process' file mode creation mask. In first release this function will be the only interface provided for initializing a process' cmask.

An initialized process cmask restricts the access permissions granted in ACDs assigned to objects created by the process. Initialized cmasks also cause ACDs to be assigned to files in MPE groups when ACDs would otherwise not be assigned. ACDs are optional for files created in MPE groups only when the creating process' cmask is uninitialized. When the creating process' cmask is uninitialized files created in MPE groups are assigned an ACD only if an ACD has been specified via the HPFOPEN ACD parameter. When

the creating process' cmask is initialized files created in MPE groups are assigned an ACD regardless of whether an ACD has been passed to HPFOPEN. MPE groups, accounts, and the root directory will not be assigned ACDs in the first release.

Once a process' file mode creation mask has been initialized, all files and hierarchical directories created by the process will be assigned ACDs containing at least the four POSIX.1 ACD entries. An initialized process cmask will also restrict the access granted in the four POSIX.1 ACD entries of ACDs specified via HPFOPEN's ACD parameter. The access granted by the four POSIX.1 entries in these ACDs will be the intersection of the access granted by the POSIX.1 ACD entries in two ACD "parameters": the HPFOPEN's ACD parameter and an ACD created from the process' cmask. The cmask ACD is created by interpreting the one's complement of the process' cmask as an ACD using the previously described mapping of the file permission bits to access permissions in the four POSIX.1 ACD entries. RACD will be granted in the ACD "parameter" created from the process' cmask. If an ACD has not been specified via the HPFOPEN ACD parameter, the cmask ACD will be assigned to the object. If the HPFOPEN ACD lacks a $OWNER entry, the cmask ACD's $OWNER entry will be added to the HPFOPEN ACD. If the HPFOPEN ACD contains entries belonging to the file group class and lacks a $GROUP_MASK entry, the cmask ACD's $GROUP_MASK entry will be added to the HPFOPEN ACD. If the HPFOPEN ACD parameter value contains a $GROUP entry, the access permissions granted by the $GROUP entry will be ANDed with the cmask ACD only if the HPFOPEN ACD contains exactly the four POSIX.1 entries. This interaction of the process cmask and ACD assignment to newly created objects will occur regardless of whether a POSIX/XL C library function or an MPE XL file system intrinsic is used to create an object.

1.6.6 Stat and Fstat

Stat() and fstat() must express access control information contained in file ACDs, directory ACDs, the file access matrix, and the alternate directory access control rules governing non-hierarchical directories as file permission bits. It is important to distinguish between *evaluating* and *reporting* access. When evaluating access to an object, all applicable access control mechanisms are enforced. When reporting access to an object, only access control information which can be expressed in the view being reported are represented. Stat() and fstat() can only return file permission bits to represent an object's access permissions. Processes should not interpret the file permission bits returned by stat()/fstat() to calculate their access to a file system object since file lockwords, file privilege level, negative file codes, the file access matrix, or an ACD may deny access represented in the file permission bits or appropriate privilege may grant access not represented in the file permission bits. In cases where the file permission bits cannot express access control governing a file system object, the file permissions bits returned by stat()/fstat() further restrict rather than weaken file system security when passed to chmod().

1.6.6.1 Reporting ACD Access

Processes must have RACD access to an ACD to successfully invoke stat() or fstat() on the object protected by the ACD. Access permissions granted by the $OWNER, $GROUP_MASK, and @.@ ACD entries are mapped to file permission bits using the previously described mapping. The file permission bits reported by stat() and fstat() are an accurate view of the underlying ACD for objects manipulated only by the POSIX/XL C library functions. These objects will have ACDs containing only the four POSIX.1 ACD entries and only access permissions which map directly to POSIX.1 access permissions. The file permission bits are incapable of accurately expressing access control information for objects protected by ACDs which do not map directly to the file permission bits.

For file ACDs stat() and fstat() map ACD R (read), W (write), and X (execute) access to POSIX.1 read, write, and execute access. ACD L (lock) and A (append) access are ignored. File owners are granted all access unless restricted by a $OWNER entry except that file execute access is handled specially. Execute access will be reported as granted to the file owner class by a file ACD when:

- The ACD contains a $OWNER entry which grants execute access.
- The ACD lacks a $OWNER entry and the file has an NMPROG, NMXL, PROG, or SL file code.
- The ACD lacks a $OWNER entry and grants execute access to at least one subject.

Write access will be reported as being granted to a process file class by a directory ACD only when *both* CD and DD access are granted to the process file class by the ACD. The file owner class is always granted create and delete directory entries access unless restricted by a $OWNER entry.

Stat() and fstat() map the access granted by the $GROUP_MASK entry to the file permission bits for the file group class. This mapping is most useful when the $GROUP_MASK represents the union of access granted by all ACD entries in the file group class. ACD operations via the ALTSEC command or the HPACDPUT intrinsic can alter an ACD so that the $GROUP_MASK ACD entry no longer represents the union of access granted by all ACD entries in the file group class. Both interfaces provide the ability to request recalculation of the $GROUP_MASK as the union of file group class ACD entry access permissions.

The file group class permission bits reported for ACDs lacking a $GROUP_MASK entry will be calculated by taking the union of the access permissions granted by all ACD entries belonging to the file group class. If there are no ACD entries belonging to the file group class, the access granted by the @.@ entry will be reported for the file group class. If the @.@ entry also doesn't exist, the file permission bits will indicate no access has been granted to the file group class.

The file permission bits will indicate no access has been granted to the file other class for ACDs lacking an @.@ entry.

1.6.6.2 Reporting File Access Matrix Access

Stat() and fstat() will ignore append and lock access when reporting access information for files protected by the file access matrix.

Stat() and fstat() will always report read and write access have been granted to the file owner class. As of MPE XL release 3.0 file owners are granted all access to files protected by the file access matrix. The CR file user type doesn't grant or deny access to file owners. Execute access will be reported as being granted to the file owner class if execute access is granted to any of the file user types or if the file has an NMXL, NMPROG, PROG, or SL file code.

Access granted to the AC file user type (which includes any access granted to the ANY file user type) will be mapped to the file group class file permission bits.

Access granted to the ANY file user type will be mapped to the file other class file permission bits.

All access (rwxrwxrwx) will be returned by stat()/fstat() for released files with NMXL, NMPROG, PROG, or SL file codes. Read/write (rw-rw-rw-) access will be returned for released files lacking these file codes.

1.6.6.3 Reporting Non-Hierarchical Directory Access

The alternate access control mechanism for MPE groups, accounts, and the root directory will cause stat()/fstat() to return file permission bits indicating directory read and execute access is granted to all file classes (r-xr-xr-x). For MPE groups, if the AC user type is granted SAVE access, then stat()/fstat() will return read/write/execute for the file group class (r-xrwxr-x).

Stat()/fstat() will report zero as the UID and GID for uninitialized directories.

1.7 SECURITY COMMANDS AND INTRINSICS

This subsection provides reference material for CI commands and MPE XL intrinsics used to manipulate file system access control information.

1.7.1 :ALTSEC Command

SYNTAX

```
ALTSEC objectname [,{FILENAME}
                    {LDEV}
                    {DEVCLASS}];

         {[ACCESS=](fileaccess[;fileaccess[;...]])}
         {NEWACD={(acdpair[;acdpair[;...]])}
                 {^filereference} }
         {COPYACD=objectname [,{FILENAME}
                               {LDEV}] }
         {ADDPAIR={(acdpair[;acdpair[;...]])}
                  {^filereference} }
         {REPPAIR={(acdpair[;acdpair[;...]])}
                  {^filereference} }
         {DELPAIR={(userspec[,userspec[,...]])}
                  {^filereference} }
         {DELACD}
         {MASK}
```

DESCRIPTION

This command changes access permissions for a file or a hierarchical directory by altering either a file access mask or an access control definition (ACD). This command does not change access permissions for MPE groups, accounts, or the root directory. Hierarchical directories do not have access masks. The file status change time stamp will be updated by successful invocations of the :ALTSEC command.

Parameter Definitions

objectname

An object name may consist of an actual file designator, a directory name, a logical device number, or a device class name. The object's type is identified by the type identifier following the objectname:

FILENAME indicates an actual file or directory designator

LDEV indicates a logical device number

DEVCLASS indicates a device class name

FILENAME is the default object type if a type identifier is not specified.

FILENAME type objectnames are interpreted using MPE-escaped semantics (see the File System ES for name semantics details.) MPE-syntax names may include wildcards and lockwords but not RFA information. Pathnames are limited to a maximum length of 255 characters (by the MYCOMMAND parameter descriptor 8-bit length field). File equations are ignored during resolution of the object name to avoid having accidental file equation references cause unintentional changes to an object's access permissions.

File lockwords must be specified for files protected by active lockwords.

A logical device number must be a numeric value and be configured on the system.

A device class name must be configured on the system.

If the FILENAME is in MPE syntax and refers to files under MPE groups, the FILENAME may contain wildcard characters.

ACCESS

Keyword which selects file access mask alteration.

fileaccess

File access mask permissions, entered as:

```
 R      ANY
 L      AC
{A} : {GU }
 W      AL
 X      GL
        CR
```

where R, L, A, W, X specify modes of access by types of users (ANY, AC, GU, AL, GL, CR) as follows:

```
R = Read
L = Lock (allows opening with dynamic locking
    option)
A = Append (implicitly specifies L also)
W = Write (implicitly specifies A and L also
X = Execute
```

Two or more modes may be specified if they are separated by commas. User types are specified as follows:

```
ANY = Any user
AC  = Member of this account only
GU  = Member of this group only
AL  = Account librarian user only
GL  = Group librarian user only
CR  = Creating user only
```

Two or more user types may be specified if they are separated by commas. Default is R, A, W, L, X: ANY.

NEWACD Keyword which selects ACD creation.

acdpair ACD's are specified in a similar manner as fileaccess. There is a modes part and a user specification part. The ACD must be in the form:

modes : userspec

NOTE: when more than one ACD pair is specified in the command line, a ";" must be used to separate them. Also, the entire ACD specification must be enclosed within parentheses (see syntax).

modes Modes is a comma-separated list of access permissions:

```
    R : READ file access
    W : WRITE file access
    L : LOCK file access
    A : APPEND file access
    X : EXECUTE file access
   CD : CREATE DIRECTORY ENTRIES access
   DD : DELETE DIRECTORY ENTRIES access
   RD : READ DIRECTORY ENTRIES access
   TD : TRAVERSE DIRECTORY ENTRIES access
 NONE : no access
 RACD : copy or read the ACD permission
```

File ACD pairs may contain R, W, L, A, X, NONE, and RACD. Directory ACD pairs may contain CD, DD, RD, TD, NONE, and RACD.

userspec The user specification is a comma-separated list of one or more of the following:

- a fully qualified user name (username.accountname)
- the file owner represented as $OWNER
- the file group represented as $GROUP
- the file group mask represented as $GROUP_MASK
- @.accountname : represents all users in the account "accountname"
- @.@ : represents all users in the system

NOTE: Wildcards cannot be used in any other manner within a user specification.

^filereference

The "^" indicates that the designated file contains the ACD definition. This is known as an indirect file. Indirect files must be ASCII files with a zero file code and a fixed record length of at most 88 bytes.

The :ALTSEC command will fail if the indirect file doesn't contain a syntactically correct ACD. ACD pairs may be on separate lines, but a pair may not span lines. Parentheses are optional when defining an "acdpair" within an indirect file.

The file reference is interpreted using MPE-escaped semantics (see the File System ES for name semantics details.) MPE-syntax names may include wildcards and lockwords but not RFA information. Pathnames are limited to a maximum length of 255 characters (by the MYCOMMAND parameter descriptor 8-bit length field). File equations are ignored during resolution of the object name to avoid having accidental file equation references cause unintentional changes to an object's access permissions.

If the file has an active lockword, it must be specified. Unqualified file names are relative to the current working directory.

COPYACD

Keyword which selects an ACD copy operation.

ADDPAIR

Keyword which selects adding one or more ACD pairs to an existing ACD.

REPPAIR

Keyword which selects replacing an ACD pair by another ACD pair.

DELPAIR

Keyword which selects deleting an ACD pair.

NOTE: Only userspec need be specified in DELPAIR.

DELACD

Keyword which selects deleting an entire ACD.

MASK

Keyword which selects recalculation of the file group class mask ($GROUP_MASK) access permissions.

USE

This command may be issued from a session, job, program, or in break. Break has no effect on this command.

OPERATION

The :ALTSEC command alters access permissions for files, hierarchical directories, logical devices, or device classes by manipulating an object's access control definition (ACD) or its access mask. Only files have access masks which can be changed using this command. An object's ACD may be altered using this command with the ACD keywords NEWACD, COPYACD, ADDPAIR, REPPAIR, DELPAIR, DELACD, and MASK. A file's access mask may be altered using either the ACCESS keyword or an access specification without a keyword. Using the ACCESS keyword is a recommended practice to help distinguish between file access mask and ACD operations.

Only a file's owner can use this command to change a file's access mask. Object owners and users with appropriate privilege can use this command to manipulate an object's ACD. Devices are owned by system managers. The ability to manipulate an ACD or file mask is not affected by the object access currently granted to a user.

File ACDs override file lockwords and the file access matrix. ACDs permit more precise access control than can be expressed using the file access matrix by allowing access permissions to be granted or denied to specific users. MPE XL allows a maximum of 40 ACD pairs to be specified for a particular object. Since a large number of ACD pair specifications will overflow the command line buffer, large numbers of ACD specifications may be entered using an indirect file.

Access to MPE groups, accounts, and the root directory cannot be altered using the :ALTSEC command. The :ALTSEC command will fail if an attempt is made to alter the access permissions for a permanent disk file whose group's home volume set is not mounted.

EXAMPLES

You have created a file named FDATA and you wish to change its file access matrix access permissions to grant write access to only yourself. Enter:

```
:ALTSEC FDATA;ACCESS=(W:CR)
```

To change the file access matrix access permissions for the FRPOG program file to allow group users to execute the program, but only account and group librarian users to read or write the file, enter:

```
:ALTSEC FPROG;ACCESS=(X:GU;R,W:AL,GL)
```

You have created a file named FDATA and you wish to assign a new ACD to FDATA granting write access to yourself and a "FRIEND" user. Enter:

```
:ALTSEC FDATA;NEWACD=(W:FRIEND.ACCT)
```

As the creator of a file, you are by default able to access the file, so granting your user identity all access in the ACD would be redundant. Users with appropriate privilege are always permitted to access files protected by ACDs.

To extend the ACD for the FDATA file so that all users on the system can read it, and all users within your account "ACCT" can also write it, enter:

```
:ALTSEC FDATA;ADDPAIR=(R:@.@; W,R:@.ACCT)
:ALTSEC FDATA;DELPAIR=(FRIEND.ACCT)
```

If you later decided that users outside your account "ACCT" should not have read access to the file FDATA any longer, enter:

```
:ALTSEC FDATA;DELPAIR=(@.@)
```

This does not mean to delete all ACD pairs, only the ACD pair matching @.@. To delete the entire ACD enter:

```
:ALTSEC FDATA;DELACD
```

You want to copy the ACD associated with ldev 5 to all devices in device class TERM:

```
:ALTSEC TERM,DEVCLASS;COPYACD=5,LDEV
```

ACDs may be copied only between objects of the same type.

You want to grant users in account ACCT all access to directory dir1:

```
:ALTSEC ./dir1;ADDPAIR=(CD,DD,RD,TD,RACD : @.ACCT)
```

You want to grant the equivalent of POSIX.1 rw-r----- file permission bits to file "a" which was created using the POSIX.1 open() function:

```
:ALTSEC ./a;REPPAIR=(RACD,R,W:$OWNER; RACD,R:$GROUP,$GROUP_MASK; NONE:@.@)
```

ERROR MESSAGES

The :ALTSEC command may display any of the CI errors associated with invalid pathname syntax (genfparse errors). Pathname syntax CI errors will be documented in a separate MPE XL CI ES. Cause and action text for the following directory traversal errors will also be specified in this separate ES:

```
Cannot traverse the pathname. (cierr934)
Non-existent directory. (cierr935)
A component of the pathname is not a directory. (cierr9027)
```

These error messages are not specified in this subsection.

The error messages specific to ALTSEC are:

```
ERRMSG : Access specification only valid for filenames. (cierr416)
CAUSE  : File access mask permissions have been specified for an object
         which is not a file.
ACTION : Use an ACD to specify access permissions.

ERRMSG : Unexpected character in file name; expected "." or "/". %
         Is the delimiter between parameters correct? . (cierr582)
CAUSE  : A pathname containing wildcard characters was specified as the
         ALTSEC objectname.
ACTION : Replace the wildcard characters in the pathname or use an MPE-syntax
         name to specify the file name pattern.

ERRMSG : Dollar character only permitted in system-defined user &
         specifications. (cierr7222)
CAUSE  : A dollar sign has been specified as a character other than the leading
         character in a user specification.
ACTION : Correct the user specification.

ERRMSG : Invalid system-defined user specification. (cierr7279)
CAUSE  : An ACD user specification starting with a dollar sign has been
         specified, but is not $OWNER, $GROUP, or $GROUP_MASK.
ACTION : Correct the user specification.

ERRMSG : System-defined user specifications are not permitted in device ACDs. &
         (cierr7280)
CAUSE  : A $OWNER, $GROUP, or $GROUP_MASK user specification has been
         specified in a device ACD.
ACTION : Remove the system-defined user specification from the device ACD
         specification.

ERRMSG : Incompatible access mode specified. (cierr7281)
CAUSE  : A file access permission has been specified for a directory or
         a directory access permission has been specified for a file.
ACTION : Correct the access permission specification.
```

```
ERRMSG : ACDs cannot be assigned to accounts, groups, or the root directory. &
         (cierr7282)
CAUSE  : The target object is an MPE group, account, or the root directory.
ACTION : Use the ALTACCT or ALTGROUP commands to change access permissions
         for accounts or MPE groups.

ERRMSG : Required ACDs cannot be deleted. (cierr7330)
CAUSE  : An attempt has been made to delete a required ACD.
ACTION : Alter the ACD using the REPPAIR or ADDPAIR keywords before removing
         ACD pairs using the DELPAIR keyword.
```

1.7.2 :RELEASE Command

SYNTAX

```
RELEASE filereference
```

DESCRIPTION

Disables file access matrix access control for a file in an MPE group. This command does not affect access control defined by lockwords or Access Control Definitions (ACDs).

Parameter Definitions

filereference Actual designator of the file whose file access matrix access control is to be disabled. Filereference is interpreted using MPE-escaped semantics (see the File System ES for name semantics details.) MPE-syntax names may include wildcards and lockwords but not RFA information. Pathnames are limited to a maximum length of 255 characters (by the MYCOMMAND parameter descriptor 8-bit length field). File equations are ignored during resolution of the object name to avoid having accidental file equation references cause unintentional changes to an object's access permissions.

If the file has a lockword, it must be specified.

USE

This command may be issued from a session, job, program, or in break. Break has no effect on this command.

OPERATION

Disables file access matrix access control for files in MPE groups. The :RELEASE command does not affect access control defined by lockwords or ACDs. If a file does not have an ACD or a lockword and is in an MPE group, disabling file access matrix access control grants all users on a system unlimited access to the file. File access matrix access control remains disabled until explicitly enabled via a :SECURE command.

This command can be successfully invoked only by a file's creator. Files outside MPE groups are protected by ACDs. The :RELEASE command will fail if the file's group's home volume set is not mounted.

EXAMPLE

To disable the file access matrix access control for the file named FILE1, enter:

```
:RELEASE FILE1
```

ERROR MESSAGES

```
RELEASE only operates on files. (cierr403)
```

1.7.3 :SECURE Command

SYNTAX

```
SECURE filereference
```

DESCRIPTION

Restores file access matrix access control for a file after this access control mechanism was disabled by a :RELEASE command. Enabling the file access matrix does not have an immediate effect on file access if the file is protected by an Access Control Definition (ACD). ACDs override the file access matrix.

Parameter Definitions

filereference Actual designator of the disc file whose security provisions are to be restored. Filereference is interpreted using MPE-escaped semantics (see the File System ES for name semantics details.) MPE-syntax names may include wildcards and lockwords but not RFA information. Pathnames are limited to a maximum length of 255 characters (by the MYCOMMAND parameter descriptor 8-bit length field). File equations are ignored during resolution of the object name to avoid having accidental file equation references cause unintentional changes to an object's access permissions.

If the file has a lockword, it must be specified.

USE

This command may be issued from a Session, Job, Program, or in BREAK. Break has no effect on this command.

OPERATION

Enables the file access matrix for a file after access control was disabled by a :RELEASE command. This command can be successfully invoked only by a file's creator. If the group's home volume set is not mounted, the :SECURE command will fail.

The :SECURE command does not affect access control defined by ACDs.

EXAMPLE

To restore the security provisions previously in effect for the file named FILE1, enter:

```
:SECURE FILE1
```

ERROR MESSAGES

```
SECURE only operates on files. (cierr357)
```

1.7.4 HPFSETOWNER Intrinsic

The new HPFSETOWNER intrinsic changes ownership and/or sharing of a file or a directory. HPFSETOWNER is similar to the POSIX. 1 chown() function provided by the POSIX/XL C library. The externals for the POSIX. 1 chown() function will be specified in a new draft of the POSIX/XL C Library Functions ES.

SYNTAX

```
                CA        I32       I32      I32    I32    I32
HPFSETOWNER(formaldesig, length, semantics, owner, group, status);
```

Parameter Definitions

formaldesig

Formal file designator -- character array (required)
The formal file designator is interpreted using the semantics specified by the semantics parameter value and may refer to a file or a directory. If the length parameter's value is -1 the file name or pathname specified must be terminated by a valid terminator character. A terminator is not needed if the name length is passed via the length parameter. The formaldesig value cannot reference a remote environment, a system-defined file, or a linked spool file. File equations are ignored when resolving this name reference. Explicitly backreferenced file names starting with an asterisk character are also not supported. The name may not contain wildcard characters or character classes.

length

Name length -- 32-bit signed integer by value (required)
Length of file name or pathname specified in formaldesig.

semantics

Name semantics -- 32-bit signed integer by value (required)
This parameter specifies which of three name semantics will be used to interpret the name passed in formaldesig:

- MPE-escaped semantics (0)
- MPE-only semantics (1)
- POSIX semantics (2)

owner

File owner -- 32-bit signed integer by value (required)
The new file owner ID (UID) to be assigned to the file or directory referenced by formaldesig. A value of -1 indicates the file owner ID should not be changed.

group

group owner -- 32-bit signed integer by value (required)

The new file group ID (GID) to be assigned to the file or directory referenced by formaldesig. A value of -1 indicates the file group ID should not be changed.

status status value -- 32-bit signed integer by reference (optional)
All nonzero status values will be returned with a file system status.subsys value (143). Status.info values are specified in the error subsection.

CAUTION

If an error or warning occurs and the status parameter was not specified, HPFSETOWNER will abort the calling process.

DISCUSSION

HPFSETOWNER permits object owners and users with appropriate privilege to change an object's UID or GID. One or both values may be changed in a single call. The file group ID (GID) of MPE groups and accounts cannot be changed.

An object's owner, its account manager(s), and system managers have different abilities to assign UID and GID values. System managers may specify any positive UID or GID value. An object's account manager may specify the UID of any user belonging to his/her account, but may only specify the GID associated his/her user ID. File owners lacking SM and AM capability cannot change a file's UID, but may change a file's or directory's GID to the GID associated with their user ID.

Changing an object's file owner ID (UID) or file group ID (GID) changes access control for that file or directory. For example, the file owner of an MPE group will have delete directory entry (DD) access and will be able to delete files and empty directories from that MPE group. File and directory owners are granted all access unless restricted by an ACD. File owners of files and hierarchical directories can also change the access permissions granted to the object. Changing an object's UID or GID also changes the file owner or file group referenced by $OWNER and $GROUP entries in the ACD associated with the object.

An ACD is automatically assigned to the file referenced by formaldesig if the file lacks an ACD and the group parameter specifies a different file group ID than the group ID associated with the account in which the file is located. The automatically assigned ACD grants RACD access to all users (RACD:@.@).

HPFSETOWNER updates the file status change time stamp when an object's UID or GID is changed.

HPFSETOWNER doesn't affect disk space accounting.

ERRORS

This subsection describes HPFSETOWNER errors. The errors detailed in the External Specifications for GENFPARSE will be returned by HPFOPEN for syntactic errors in file names and pathnames. None of the errors described below are unique to HPFSETOWNER. All of these status values belong to the file system subsystem (143). Existing error message text is shown as ERRMSG text. The ES TEXT, ES

CAUSE, ES ACTION, and NOTES text provide information more specific to HPFSETOWNER. This additional text is not intended for inclusion in user manuals.

```
STATUS.INFO     : -16
MNEMONIC        : IS_INVALID_OPERATION
ERRMSG          : Intrinsic layer; an invalid operation for this device type.
ES TEXT         : Invalid operation.
ES CAUSE        : The object referenced by formaldesig is an MPE group or
                  account and a value other than -1 has been specified for the
                  group parameter.  The object is a linked spool file or
                  RFA file.
ES ACTION       : The file group ID of MPE groups and accounts cannot be changed.
                  Neither the UID nor the GID of linked spool files or RFA files
                  can be changed.  Correct the formaldesig value if it is not
                  referencing the intended object.

STATUS.INFO     : -21
MNEMONIC        : IS_SECURITY_VIOLATION
ERRMSG          : Intrinsic layer; a SECURITY VIOLATION on file occurred.
ES TEXT         : Security violation.
ES CAUSE        : One of the following situations has occurred:
                    1) The user is not a system manager, an account manager, or
                       the owner of the object referenced by formaldesig.
                    2) The object's account manager has specified a UID which is
                       not associated with the manager's account.
                    3) The object's owner has specified a UID value other than -1
                       and is not the object's account manager or a system manager.
                    4) The object's account manager or owner has specified a GID
                       not associated with his/her user ID and is not a system
                       manager.
ES ACTION       : Determine which restriction is being violated and respecify
                  the values being passed to HPFSETOWNER.
```

```
STATUS.INFO    : -44
MNEMONIC       : IS_DIRECTORY_IO_ERROR
ERRMSG         : Intrinsic layer; the operation could not be completed because
                 an I/O error occurred in the directory routines.
ES TEXT        : Directory I/O error.
ES CAUSE       : A directory I/O error occurred.
ES ACTION      : If directory I/O errors persist, contact your HP service
                 representative or schedule system down time in which to
                 recover the volume.

STATUS.INFO    : -61
MNEMONIC       : IS_INTERNAL_ERROR
ES TEXT        : Internal file system error.
ES CAUSE       : An internal error occurred in the file system or a subsystem
                 called by the file system.
ES ACTION      : Contact your HP support representative.
NOTES          : Several errors are folded into this status value:
                 Failure to start or end a label transaction.
                 Failure to start or finish a label write.
                 Failure to SM_OPEN or SM_CLOSE the object.

STATUS.INFO    : -65
MNEMONIC       : IS_UID_GID_DATA_BASE_NOT_FOUND
TEXT           : UID-GID database not found.
CAUSE          : The UID/GID database could not be opened.
ACTION         : The system manager can run the PXUTIL.PUB.SYS utility
                 to create a UID/GID database.
NOTES          : This error may be returned as long as we have to convert
                 numeric UID/GID parameter values to string values stored
                 in the file label.

STATUS.INFO    : -66
MNEMONIC       : IS_UID_NOT_FOUND
TEXT           : User information not found.
CAUSE          : File owner information was not found in the UID database
                 because the database is out of sync with the directory or the
                 user is no longer defined on the system.
ACTION         : No action is required if the user ID has been explicitly removed
                 from the system.  If the user ID should exist on the system,
                 the system manager can run the utility PXUTIL.PUB.SYS to
                 synchronize the UID/GID database with the directory.
NOTES          : This error may be returned as long as we have to convert
                 numeric UID/GID parameter values to string values stored
                 in the file label.
```

```
STATUS.INFO   : -67
MNEMONIC      : IS_GID_NOT_FOUND
TEXT          : Group information not found.
CAUSE         : File group information was not found in the GID database
                because the database is out of sync with the directory or
                the group is no longer defined on the system.
ACTION        : No action is required if the group has been explicitly removed
                from the system.  If the group should exist on the system,
                the system manager can run the utility PXUTIL.PUB.SYS to
                synchronize the UID/GID database with the directory.
NOTES         : This error may be returned as long as we have to convert
                numeric UID/GID parameter values to string values stored
                in the file label.

STATUS.INFO   : -69
MNEMONIC      : IS_COMPONENT_IN_PATHNAME_NOT_A_DIRECTORY
TEXT          : Component of pathname is not a directory.
CAUSE         : A pathname component does not exist or a name component
                other than the last name component referenced an object which
                is not a directory.
ACTION        : Correct the pathname.

STATUS.INFO   : -247
MNEMONIC      : HOP_FILE_NOT_FOUND
ERRMSG        : The file does not exist in specified domain; the file was not
                found. (FILE OPEN ERROR -247)
ES CAUSE      : A directory entry corresponding to the file name component
                was not found.
ES ACTION     : Use listfile to inspect the file name or pathname and correct
                the erroneous name component.

STATUS.INFO   : -456
MNEMONIC      : HOP_PREVIOUS_EXCLUSIVE
ERRMSG        : The file could not be opened because the file is being stored,
                or is opened in a way that does not allow additional write
                access.  (FILE OPEN ERROR -456)
ES TEXT       : File in use.
ES CAUSE      : File is a store or restore candidate.
ES ACTION     : Retry the HPFSETOWNER call after the store or restore is
                finished.

STATUS.INFO   : -1036
MNEMONIC      : TM_HPDIR_NO_TRAVERSE_ACCESS
TEXT          : User lacks traverse directory entry (TD) access for a directory.
CAUSE         : A directory ACD has not granted the user TD access to a
                directory.
ACTION        : Determine why the user has not been granted TD access.
                Have someone with sufficient access grant TD access if
                granting access is appropriate.
```

1.7.5 HPACDINFO Intrinsic

The HPACDINFO intrinsic queries information about an ACD associated with a target file, directory, or device.

SYNTAX

```
                I32        I16V         *
HPACDINFO(status, target_type, target
                  [,info_type1, info1]
                  [,info_type2, info2]
                  [,info_type3, info3]
                  [,info_type4, info4]);
```

HPACDINFO is similar to HPFOPEN in its use of (type, value) parameter pairs to specify its arguments. The type parameter indicates how to interpret the value of its associated parameter. The (target_type, target) parameter pair indicates the file, directory, or device from whose ACD information is to be queried. The other (info_type, info) parameter pairs indicate the ACD information to be returned. Status values are documented in the MPE XL FOS Security ES. Pathnames are accepted only when POSIX is enabled.

The following target_types are may be specified:

- HGS_OPTION_FILE_NAME - indicates the target parameter is the name of a target file or directory whose ACD will be queried. The target parameter is a byte array containing the target file name. The target name is interpreted using MPE-escaped semantics. MPE-syntax names may contain a lockword, but may not contain wildcards or RFA information. MPE-syntax names shorter than 35 characters must be terminated by a carriage return, a blank, or a null character. Pathnames must be terminated by a null character.

- HGS_OPTION_PATHNAME - indicates the target parameter is the pathname of a target file or directory whose ACD will be queried. The target parameter is a byte array containing the target pathname. Pathnames must be terminated by a null character.

- HGS_OPTION_UFID - indicates the target parameter is a 20-byte unique file identifier (UFID) structure. UFIDs can be obtained from the FLABELINFO and FFILEINFO file system intrinsics.

- HGS_OPTION_FILE_NUM - indicates the target parameter is a 16-bit MPE file number of an open file or directory.

- HGS_OPTION_DEVICE_NUM - indicates the target parameter is a 16-bit logical device number (ldev).

- HGS_OPTION_DEVICE_NAME - indicates the target parameter is an 8-byte name of a specific device.

The following info_type parameters may be specified without selecting an ACD pair using the HGS_OPTION_USER_SPEC_IN info_type:

- HGS_OPTION_GET_VERSION - requests the ACD's version identifier to be returned as a 16-bit integer in the associated info parameter.
- HGS_OPTION_GET_NUM_ENTRIES - requests the number of entries in the ACD to be returned as a 16-bit integer in the associated info parameter.
- HGS_OPTION_GET_FIRST_USER - requests the user specification from the first ACD entry to be returned as an 18-character byte array in the associated info parameter.

The following info_type parameters request information from an ACD pair. These info_type parameters require an ACD pair to be selected using the HGS_OPTION_USER_SPEC_IN info_type. Only one ACD pair may be selected per HPACDINFO call. If more than one HGS_OPTION_USER_SPEC_IN info_type parameter is specified, the last parameter is used to select the ACD pair. The info_type parameters for a specific ACD pair are:

- HGS_OPTION_USER_SPEC_IN - indicates the associated info parameter is an 18-character byte array specifying the user whose ACD pair is to be queried for information using one or more of the following info_type values.
- HGS_OPTION_GET_MODES_ASCII - requests the ACD modes from the selected pair to be returned in ASCII format as a 25-character byte array.
- HGS_OPTION_GET_MODES_BIT - requests the ACD modes from the selected pair to be returned as a bit mask in a 16-bit integer:

  R (0:1)
  W (1:1)
  X (2:1)
  A (3:1)
  L (4:1)
  Z (8:1) - Permission to read ACD.
  TD (11:1)
  RD (12:1)
  CD (13:1)
  DD (14:1)
  N (15:1) - No access.

- HGS_OPTION_GET_MODES_EVALUATED_ASCII - requests that the access modes granted to the selected ACD pair be returned in ASCII format as a 25-character byte array. The access granted to the user specified in the selected ACD pair may be greater than the access specified in the ACD pair when the user has appropriate privilege.
- HGS_OPTION_GET_MODES_EVALUATED_BIT - requests that the access modes granted in the selected ACD pair be returned as a bit mask in a 16-bit integer. This bit mask has the same definition previously defined for the HGS_OPTION_GET_MODES_BIT

info_type value. The access granted to the user specified in the selected ACD pair may be greater than the access specified in the ACD pair when the user has appropriate privilege.

- HGS_OPTION_GET_NEXT_USER - requests the user specification from the next ACD pair to be returned as an 18-character array. This information can be used in a subsequent HPACDINFO call in combination with the HGS_OPTION_USER_SPEC_IN info_type to query information about the next ACD pair.

HPACDINFO Type Constant Values

The values for the previously described parameter type constants are:

```
HGS_OPTION_FILE_NAME                = 1
HGS_OPTION_UFID                     = 3
HGS_OPTION_FILE_NUM                 = 4
HGS_OPTION_DEVICE_NUM               = 6
HGS_OPTION_DEVICE_NAME              = 7
HGS_OPTION_PATHNAME                 = 11
HGS_OPTION_GET_VERSION              = 20
HGS_OPTION_GET_NUM_ENTRIES          = 21
HGS_OPTION_GET_FIRST_USER           = 22
HGS_OPTION_USER_SPEC_IN             = 30
HGS_OPTION_GET_MODES_ASCII          = 31
HGS_OPTION_GET_MODES_BIT            = 32
HGS_OPTION_GET_MODES_EVALUATED_ASCII = 33
HGS_OPTION_GET_MODES_EVALUATED_BIT  = 34
HGS_OPTION_GET_NEXT_USER            = 35
```

Capabilities Required

Any user with RACD access to an ACD may use HPACDINFO to get information about that ACD.

EXAMPLE

This example illustrates how HPACDINFO could be used to obtain the number of entries in the ACD and the user specification from the first pair in the ACD associated with the file TARGET:

```
program acdinfo (input, output);

type
  shortint = -32768..32767;

const HGS_OPTION_FILE_NAME = 1,
      HGS_OPTION_GET_NUM_ENTRIES = 21,
      HGS_OPTION_GET_FIRST_USER = 22;

var
  status      : integer;
  target_file : packed array [1..28] of char;
  numentry    : shortint;
  first_user  : packed array [1..18] of char;

procedure HPACDINFO;intrinsic;

begin
target_file := 'TARGET ';
HPACDINFO(status,
          HGS_OPTION_FILE_NAME, target_file,
          HGS_OPTION_GET_NUM_ENTRIES, numentry,
          HGS_OPTION_GET_FIRST_USER, firstuser);
if status = 0
   then
      begin
      writeln('Number of Entries: ', numentry:1);
      writeln('First UserSpec: ', firstuser);
      end
   else writeln('Error in invocation of HPACDINFO');

end.
```

1.7.6 HPACDPUT Intrinsic

The HPACDPUT intrinsic manipulates ACD information associated with a file, directory, or device. ACD changes update the file status change time stamp.

SYNTAX

```
                I32        I16V         *
HPACDPUT (status, target_type, target,
                   ACD_operation, ACD_value);
```

HPACDPUT is similar to HPFOPEN in its use of (type, value) parameter pairs to specify its arguments. The type parameter indicates how to interpret the value of its associated parameter. The (target_type, target) parameter pair specifies the file system object whose ACD is to be manipulated. The (ACD_operation, ACD_info) parameter pair specifies the ACD operation to be performed. Status values are documented in the MPE XL FOS Security ES. Pathnames are accepted only when POSIX is enabled.

The target_type parameter may be one of the following values:

- HPS_OPTION_FILE_NAME - indicates the target parameter is the name of a target file or directory whose ACD will be manipulated. The target parameter is a byte array containing the target file name. The target name is interpreted using MPE-escaped semantics. MPE-syntax names may contain a lockword, but may not contain wildcards or RFA information. MPE-syntax names shorter than 35 characters must be terminated by a carriage return, a blank, or null character. Pathnames must be terminated by a null character.

- HPS_OPTION_PATHNAME - indicates the target parameter is the pathname of a target file or directory whose ACD will be manipulated. The target parameter is a byte array containing the target pathname. Pathnames must be terminated by a null character.

- HPS_OPTION_GENERIC_FILE_NAME - indicates the target parameter is an MPE-only file name which may contain MPE wildcard characters (@, #, and ?). The target parameter is a byte array with a maximum length of 35 characters. Generic file names may not contain lockwords.

- HPS_OPTION_UFID - indicates the target parameter is a 20-byte unique file identifier (UFID) structure. UFIDs can be obtained from the FLABELINFO and FFILEINFO file system intrinsics.

- HPS_OPTION_FILE_NUM - indicates the target parameter is a 16-bit MPE file number of an open file or directory.

- HPS_OPTION_ALL_DEVICES - indicates all devices configured on the system are the target of the ACD operation. This target_type does not require a target parameter to be specified.

- HPS_OPTION_DEVICE_NUM - indicates the target parameter is a 16-bit logical device number (ldev).

- HPS_OPTION_DEVICE_NAME - indicates the target parameter is an 8-byte name of a specific device.

- HPS_OPTION_DEVICE_CLASS - indicates the target parameter is a byte array of up to 8 characters specifying the name of a device class.

The following (ACD_operation, ACD_value) pairs describe both the ACD operation to be performed and the source of the ACD.

- HPS_OPTION_NEW_ACD_FROM_PARM - the ACD_value parameter is a byte array with a maximum length of 256 characters specifying the new ACD to be assigned to the target file, directory, or device.

- HPS_OPTION_ADD_PAIRS_TO_ACD_FROM_PARM - the ACD_value parameter is a byte array with a maximum length of 256 characters specifying the ACD pairs to be added to the target ACD.

- HPS_OPTION_REPLACE_PAIRS_IN_ACD_FROM_PARM - the ACD_value parameter is a byte array with a maximum length of 256 characters specifying ACD pairs to be replaced in the target ACD.

- HPS_OPTION_DELETE_PAIRS_IN_ACD_FROM_PARM - the ACD_value parameter is a byte array with a maximum length of 256 characters specifying ACD pairs to be deleted from the target ACD.

- HPS_OPTION_DELETE_ACD - delete the entire ACD. No ACD_value parameter is required with this operation_type.

- HPS_OPTION_NEW_ACD_FROM_FILE - the ACD_value parameter is a byte array specifying the name of an indirect file containing the new ACD to assign to the target file, directory, or device. The indirect file name is interpreted using MPE-escaped semantics. MPE-syntax names may contain a lockword, but may not contain wildcards or RFA information. MPE-syntax names shorter than 35 characters must be terminated by a carriage return, a blank, or null character. Pathnames must be terminated by a null character.

- HPS_OPTION_COPY_ACD_FROM_FILE - the ACD_value parameter is a byte array specifying the name of a file or directory whose ACD will be copied to the target file/directory. The name of the file/directory whose ACD will be copied does not need to be fully qualified and may be expressed using either MPE or POSIX name syntax. POSIX pathnames must start with a dot or slash character and may be at most {PATH_MAX} bytes long including the leading dot or slash and a terminating null character. MPE-syntax names shorter than 35 characters must be terminated by a carriage return, blank, or null character. ACDs may be copied only between objects of the same type.

- HPS_OPTION_COPY_ACD_FROM_UFID - the ACD_value parameter is a 20-byte array specifying the UFID of a file or directory whose ACD will be copied to the target file/directory. ACDs may be copied only between objects of the same type.

- HPS_OPTION_COPY_ACD_FROM_FILENUM - the ACD_value parameter is a 16-bit integer specifying the MPE file number of a file or directory whose ACD will be copied to the target file/directory. ACDs may be copied only between objects of the same type.

- HPS_OPTION_COPY_ACD_FROM_DEV_NAME - the ACD_value parameter is an 8-byte array specifying the name of a device whose ACD will be copied to the target device. ACDs may be copied only between objects of the same type.

- HPS_OPTION_ADD_PAIRS_TO_ACD_FROM_FILE - the ACD_value parameter is a byte array specifying the name of an indirect file containing ACD pairs to be added to the target file, directory, or device ACD. The indirect file name is interpreted using MPE-escaped semantics. MPE-syntax names may contain a lockword, but may not contain wildcards or RFA information. MPE-syntax names shorter than 35 characters must be terminated by a carriage return, a blank, or null character. Pathnames must be terminated by a null character.

- HPS_OPTION_REPLACE_PAIRS_IN_ACD_FROM_FILE - the ACD_value parameter is a byte array specifying the name of an indirect file containing ACD pairs to be replaced in the target file, directory, or device ACD. The indirect file name is interpreted using MPE-escaped semantics. MPE-syntax names may contain a lockword, but may not contain wildcards or RFA information. MPE-syntax names shorter than 35 characters must be terminated by a carriage return, a blank, or null character. Pathnames must be terminated by a null character.

- HPS_OPTION_DELETE_PAIRS_IN_ACD_FROM_FILE - the ACD_value parameter is a byte array specifying the name of an indirect file containing ACD pairs to be deleted from the target file, directory, or device ACD. The indirect file name is interpreted using MPE-escaped semantics. MPE-syntax names may contain a lockword, but may not contain wildcards or RFA information. MPE-syntax names shorter than 35 characters must be terminated by a carriage return, a blank, or null character. Pathnames must be terminated by a null character.

- HPS_OPTION_MERGE_ACD_PAIRS_FROM_PARM - the ACD_value parameter is a byte array with a maximum length of 256 characters specifying the ACD pairs to be merged with the target ACD. Merging two ACDs is a combination of replacing and adding ACD pairs in the target ACD. Pairs which occur in both the source and target ACDs are replaced. Pairs which occur only in the source ACD are added to the target ADD.

- HPS_OPTION_MERGE_ACD_PAIRS_FROM_FILE - the ACD_value parameter is a byte array specifying the name of an indirect file containing ACD pairs to be merged with the target file, directory, or device ACD. The indirect file name is interpreted using MPE-escaped semantics. MPE-syntax names may contain a lockword, but may not contain wildcards or RFA information. MPE-syntax names shorter than 35 characters must be terminated by a carriage return, a blank, or null character. Pathnames must be terminated by a null character.

- HPS_OPTION_CALCULATE_MASK - the ACD_value parameter is a 32-bit integer indicating the file class whose maximum access permissions are to be calculated and assigned to the class mask. The constant HPS_GROUP_MASK will be used as the operation value to request a recalculation of the file group class mask. The group class mask is the only class mask in the first release. The class mask is added to the ACD if it does not already exist.

HPACDPUT Type Constants

The values for the previously described HPACDPUT type constants are:

```
HPS_OPTION_FILE_NAME                          1
HPS_OPTION_GENERIC_FILE_NAME                  2
HPS_OPTION_UFID                               3
HPS_OPTION_FILE_NUM                           4
HPS_OPTION_ALL_DEVICES                        5
HPS_OPTION_DEVICE_NUM                         6
HPS_OPTION_DEVICE_NAME                        7
HPS_OPTION_DEVICE_CLASS                       8
HPS_OPTION_PATHNAME                          11
HPS_OPTION_NEW_ACD_FROM_PARM                 20
HPS_OPTION_ADD_PAIRS_TO_ACD_FROM_PARM        21
HPS_OPTION_REPLACE_PAIRS_IN_ACD_FROM_PARM    22
HPS_OPTION_DELETE_PAIRS_IN_ACD_FROM_PARM     23
HPS_OPTION_DELETE_ACD                        24
HPS_OPTION_NEW_ACD_FROM_FILE                 25
HPS_OPTION_COPY_ACD_FROM_FILE                26
HPS_OPTION_COPY_ACD_FROM_UFID                27
HPS_OPTION_COPY_ACD_FROM_FILENUM             28
HPS_OPTION_COPY_ACD_FROM_DEV_NUM             29
HPS_OPTION_COPY_ACD_FROM_DEV_NAME            30
HPS_OPTION_ADD_PAIRS_TO_ACD_FROM_FILE        31
HPS_OPTION_REPLACE_PAIRS_IN_ACD_FROM_FILE    32
HPS_OPTION_DELETE_PAIRS_IN_ACD_FROM_FILE     33
HPS_OPTION_MERGE_ACD_PAIRS_FROM_PARM         34
HPS_OPTION_MERGE_ACD_PAIRS_FROM_FILE         35
HPS_OPTION_CALCULATE_MASK                    50
HPS_GROUP_MASK                                1
```

Capabilities Required:

ACDs may only be manipulated by an object's file owner or by processes with appropriate privilege. Device ACDs may only be manipulated by processes with SM capability.

EXAMPLE

This example illustrates how HPACDPUT could be used to create an ACD for the file TARGET:

```
program acdput (input, output);

const
   HPS_OPTION_FILE_NAME = 1,
   HPS_OPTION_NEW_ACD_FROM_PARM = 20;

var
  status   : integer;
  filename : packed array [1..28] of char;
  ACD      : packed array [1..256] of char;

procedure HPACDPUT;intrinsic;

begin
filename := 'TARGET ';
ACD      := '(x:@.@;r,w:mgr.sys)';
ACD[20]  := #m;
HPACDPUT(status,
         HPS_OPTION_FILE_NAME, filename,
         HPS_OPTION_NEW_ACD_FROM_PARM, ACD);

if status <> 0
   then writeln('HPACDPUT failed.  Status = ', status);
end.
```

1.8 PHASE ONE RESTRICTIONS

Chmod() will not be able to assign ACDs to MPE groups, accounts, or the root directory.

Supplemental GIDs will not be supported.

Processes will not be able to change their user identity. Setuid() and setgid() will not be implemented in phase I.

File owner and file group will be string-based.

File lockwords, privilege levels, negative file codes, and write protection may only be used with files in MPE groups.

HPFSETOWNER will not change MPE group or account GIDs.