PSSG Draft 17

STANDARDS PROJECT

Draft Standard for Information Technology— Portable Operating System Interface (POSIX)— Part 2: Shell and Utilities— Amendment #: Protection and Control Interfaces

Sponsor

Technical Committee on Operating Systems
and Application Environments
of the
IEEE Computer Society

Work Item Number:
JTC1 22.43 %

Abstract: IEEE Std 1003.2c is an amendment to IEEE Std 1003.2-1992. It |
defines security utilities to open systems for access control lists, separation of +
privilege (capabilities), mandatory access control, and information label mechan- +
isms.

Keywords: access control lists, application portability, information labels, mandatory access control, capability, open systems, operating systems, portable application, POSIX, POSIX.2, privilege, security, user portability +

PSSG / D17
October 1997

ISBN-xxxx-xxxxx-x

Library of Congress Catalog Number 90-xxxxx

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
October 1997 *XXXXXXX*

Foreword

NOTE: This foreword is not a normative part of the standard and is included for informative purposes only.

The purpose of this standard is to define a standard interface and environment for Computer Operating Systems that require a secure environment. The standard is intended for system implementors and application software developers. It is an extension to the IEEE P1003.2 (POSIX.2).

Organization of the Standard

The standard is divided into several parts:

- — Revisions to the General Section (Section 1)
- — Revisions to Terminology and General Requirements (Section 2)
- — Revisions to Execution Environment Utilities (Section 4)
- — Revisions to User Portability Utilities (Section 5)
- — Access Control Lists (Section 8)
- — Capability (Section 9) %
- — Mandatory Access Control (Section 10) %
- — Information Labeling (Section 11)
- — Annex E - Revisions to Rationale and Notes
- — Annex I - Ballot Instructions

Changes to the draft since the previous ballot are indicated by one of four marks %
in the right-hand margin. These change marks should aid the balloter in deter- %
mining what has changed and therefore what is candidate text for comments and %
objections during this ballot. A bar ("|") indicates changes to the line between %
drafts 15 and 16. A plus ("+") indicates that text has been added in draft 16. A %
minus ("-") indicates that text present in that location in draft 15 has been deleted %
in draft 16. A percent ("%") indicates that a change was made at that location in %
draft 17.

Conformance Measurement

In publishing this standard, both IEEE and the security working group simply intend to provide a yardstick against which various operating system implementations can be measured for conformance. It is not the intent of either IEEE or the security working group to measure or rate any products, to reward or sanction any vendors of products for conformance or lack of conformance to this standard, or to attempt to enforce this standard by these or any other means. The responsibility for determining the degree of conformance or lack thereof with this standard rests solely with the individual who is evaluating the product claiming to be in conformance with this standard.

Extensions and Supplements to This Standard

Activities to extend this standard to address additional requirements can be anticipated in the future. This is an outline of how these extensions will be incorporated, and also how users of this document can keep track of that status. Extensions are approved as "Supplements" to this document, following the IEEE Standards Procedures. Approved Supplements are published separately and are obtained from the IEEE with orders for this document until the full document is reprinted and such supplements are incorporated in their proper positions.

If you have any questions regarding this or other POSIX documents, you may contact the the IEEE Standards Office by calling IEEE at:

1 (800) 678-IEEE - from within the US
1+ (908) 981-1393 from outside the US

to determine what supplements have been published. Published supplements are available for a modest fee.

Supplements are numbered in the same format as the main document with unique positions as either subsections or main sections. A supplement may include new subsections in various sections of the main document as well as new main sections. Supplements may include new sections in already approved supplements. However, the overall numbering shall be unique so that two supplements only use the same numbers when one replaces the other. Supplements may contain either required or optional facilities. Supplements may add additional conformance requirements (see POSIX.2, Implementation Conformance, 1.3) defining new classes of conforming systems or applications.

It is desirable, but perhaps not avoidable, that supplements do not change the functionality of the already defined facilities. Supplements are not used to provide a general update of the standard. A general update of the standard is done through the review procedure as specified by the IEEE.

If you have interest in participating in Portable Applications Standards Committee (PASC) working groups please send your name, address and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, and ask to have your request forwarded to the chairperson of the appropriate PASC working group. If you have interest in participating in this work at the international level, contact your ISO/IEC national body.

Please report typographical errors and editorial changes for this draft standard directly to:

Casey Schaufler
Silicon Graphics
2011 North Shoreline Blvd.
P.O. Box 7311
Mountain View, CA 94039-7311
(415) 933-1634 (voice)
(415) 962-8404 (fax)
casey@sgi.com
Schaufler@DOCKMASTER.NCSC.MIL

IEEE Std 1003.2c was prepared by the security Working Group, sponsored by the Portable Applications Standards Committee (PASC) of the IEEE Computer Society.

Standards Subcommittee for PASC

Chair:	Lowell Johnson
Treasurer:	Barry Needham
Secretary:	Charles Severence

1003.6 Working Group Officials

Chair:	Lynne Ambuel
Technical Editor:	Casey Schaufler

The following people have participated in the security Working Group.

Lynne Ambuel
Jeanne Baccash
Lee Badger
Martin Bailey
John-Olaf Bauner
D. Elliott Bell
Lowell Bogard
Kevin Brady
Joe Brame
Matthew Brisse
Joseph Bulger
Lisa Carnahan
Mark Carson
Charisse Castagnoli
Paul Close
Roland Clouse
Peter E. Cordsen
Janet Cugini
Anthony D'Alessandro
Daniel D. Daugherty
Manilal Daya
Ana Maria De Alvare'
Terence Dowling
Jack Dwyer
Maryland R. Edwards
Ron Elliott
Lloyd English
Jeremy Epstein
Frank Fadden
Kevin Fall
David Ferbrache
Carl Freeman
Mark Funkenhauser
Morrie Gasser
Gerald B. Green
John Griffith
Henry Hall
Craig Heath
Tom Houghton
Rand Hoven
Chris Hughes
Howard Israel
Paul A. Karger
Joseph Keenan
Jerry Keselman
Yvon Klein
Andy Kochis
Steve Kramer
Steven LaFountain
Danielle Lahmani
Jason Levitt
Warren E. Loper
Jeff Mainville
Doug Mansur
Richard E. Mcnaney
Chris Milsom
Mark Modig
Jim Moseman
Kevin V. Murphy
Greg Nuss
Rose Odonnell
Gary Oing
C. Larry Parker
Gordon Parry
Jeff Picciotto
Michael Ressler
David Rogers
Peter L. Rosencrantz
Shawn Rovansek
Craig Rubin
Roman Saucedo
Stuart Schaeffer
Mark Schaffer
Casey Schaufler
Michael Schmitz
Larry Scott
Eric Shaffer
Olin Sibert
Rick Siebenaler
Alan Silverstein
Jon Spencer
Dennis Steinauer
Chris Steinbroner
Michael Steuerwalt
Doug Steves
Steve Sutton
W. Lee Terrell
Charlie Testa
Jeff Tofano
Brian Weis
Catherine West
Ken Witte

Information technology—Portable operating system interface for computer environments

Section 1: Revisions to the General Section

⇒ **1.1 Scope** *This scope is to be revised and integrated appropriately into the scope when POSIX.2c is approved:*

This standard, P1003.2c/D17: October 1997 (POSIX.2c), defines four independent, security-related, optional sets of utilities. These interfaces will provide changes and additions to ISO/IEC 9945-2 (Shell and Utilities) as they are published and approved. The sets of utilities for implementation are:

(1) Access Control Lists (ACL)

(2) Capability

(3) Mandatory Access Controls (MAC)

(4) Information Labeling (IL)

Each option defines new utilities, as well as security-related constraints for the functions and utilities defined by other POSIX standards.

⇒ **1.2 Normative References (POSIX.2: line 92)** *Modify normative reference 8 (IEEE Std 1003.1-1990) to refer to POSIX.1 as amended by P1003.1e.* %

⇒ **1.3.1.3 Conforming Implementation Options (POSIX.2: line 172)** *Insert the following options in alphabetic order:*

{POSIX2_ACL} The system supports the Access Control List Utilities Option (see Section 8). |

{POSIX2_CAP} The system supports the Capability Utilities Option (see Section 9). %

{POSIX2_INF} The system supports the Information Label Utilities Option (see Section 11). %

{POSIX2_MAC} The system supports the Mandatory Access Control Utilities Option (see Section 10). %

Section 2: Revisions to Terminology and General Requirements

⇒ 2.2 Definitions

⇒ **2.2.2 General Terms** *Delete 2.2.2.66 file access permissions Modify the contents of subclause 2.2.2, General Terms, to add or modify the indicated definitions in the correct sorted order [disregarding the subclause numbers shown here.]*

2.2.2.1 access ACL: An access control list (ACL) which is used in making discretionary access control decisions for an object. [POSIX.1e]

2.2.2.2 access control: The prevention of unauthorized access to objects by processes and, conversely, the permitting of authorized access to objects by processes. [POSIX.1e]

2.2.2.3 access control list (ACL): A discretionary access control entity associated with an object, consisting of a list of entries where each entry is a user identifier coupled with a set of access permissions. [POSIX.1e]

2.2.2.4 capability: An attribute of a process that determines whether or not a process has the appropriate privilege to perform a specific POSIX.1 action where appropriate privilege is required. [POSIX.1e]

2.2.2.5 capability state: A grouping of all of the flags defined by an implementation for a capability. [POSIX.1e]

2.2.2.6 default ACL: An ACL which is used in determining the initial discretionary access control information for objects created. [POSIX.1e]

2.2.2.7 discretionary access control (DAC): A means of determining and enforcing access to objects based on the identity of the user, process, and/or groups to which the objects belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to other subjects. [POSIX.1e]

2.2.2.8 file access controls: One standard file access control mechanism based on file permission bits and two optional file access control mechanisms, based on access control lists and mandatory access control labels, are defined by this standard.

2.2.2.8.1 file access permissions

This standard defines discretionary file access control on the basis of file permission bits as described below. The additional provision, access control lists, applies only if {_POSIX2_ACL} is defined. The additional provision, mandatory access control, applies only if {_POSIX2_MAC} is defined.

The file permission bits of a file contain read, write, and execute/search permissions for a file owner class, file group class, and file other class.

Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An additional access control mechanism shall only further restrict the access permissions defined by the file access control mechanisms described in this section. An alternate access control mechanism shall:

(1) Specify file permission bits for the file owner class, file group class, and file other class corresponding to the access permissions.

(2) Be enabled only by explicit user action, on a per user basis by the file owner or a user with the appropriate privilege.

(3) Be disabled for a file after the file permission bits are changed for that file with the `chmod` utility. The disabling of the alternate mechanism need not disable any additional mechanisms defined by an implementation.

Whenever a process requests file access permission for read, write, or execute/search, if no additional mechanism denies access, access is determined as follows:

If the process possesses appropriate privilege:

— If read, write, or directory search permission is requested, access is granted.

— If execute permission is requested, access is granted if execute permission is granted to at least one user by the file access permission bits.

Otherwise: Access is granted on the basis of the evaluation of the file permission bits.

⇒ **2.2.2.8.2 access control lists:** *Add this as a new concept.*

The {_POSIX2_ACL} option provides an additional access control mechanism by providing file access control based upon an access control list mechanism. The additional provisions of this subclause apply only if {_POSIX2_ACL} is defined. The interaction between file permission bits and the ACL mechanism is defined such that a correspondence is maintained between them. The ACL mechanism therefore enhances access control based upon the file permission bits.

An ACL entry shall support at a minimum read, write, and execute/search permissions.

An ACL is set at file creation time. An additional *default ACL* can be associated with a directory; this is used in setting the ACL of any object created in that directory.

Each access mode requested shall be individually evaluated against the ACL. A process is granted discretionary access to a file only if all individual requested modes of access are granted or the process possesses appropriate privileges.

If the process possesses appropriate privilege:

— If read, write or directory search permission is requested, access is granted.

— If execute permission is requested, access is granted if execute permission is specified in at least one ACL entry.

Otherwise, access is granted on the basis of the evaluation of the ACL permissions.

⇒ **2.2.2.8.3 mandatory access control:** *Add this as a new concept.*

The {_POSIX2_MAC} option provides utilities to an additional access control mechanism based on the assignment of MAC labels to subjects and objects. The provisions of this subclause only apply if {_POSIX2_MAC} is defined.

The MAC mechanism permits or restricts access to an object by a process based on a comparison of the MAC label of the process to the MAC label of the object. A process can read an object if the process's MAC label dominates the object's MAC label, and write an object if the process's MAC label is dominated by the object's MAC label. However, an implementation may impose further restrictions, permitting write access to objects only by processes with a MAC label equivalent to that of the object. This standard does not define the dominance and equivalence relationships, and thus does not define a particular MAC policy.

MAC read access to an object by a process requires that the process's MAC label dominate the object's MAC label or that the process possess appropriate privilege.

MAC write access to an object by a process requires that the process's MAC label be dominated by the object's MAC label or that the process possess appropriate privilege.

Execute/search file access requires MAC read access to the file.

The MAC label of an object (including a process object) is set at creation time to dominate the MAC label of the creating process. Although this allows creation of upgraded objects, this standard provides only interfaces which will create objects with MAC labels equivalent to that of the creating process. However, interfaces are provided to allow an appropriately privileged process to upgrade existing objects.

⇒ **2.2.2.8.4 evaluation of file access:** *Add this as a new concept.*

Whenever a process requests file access, if no alternate access control mechanism applies, then access shall be granted only if all the applicable POSIX.1 access control mechanisms and any additional access control mechanisms grant the access.

2.2.2.9 dominate: An implementation-defined relation between the values of MAC labels or implementation labels. [POSIX.1e]

2.2.2.10 file group class: The property of a file indicating access permissions for a process related to the process's group identification.

A process is in the file group class of a file if the process is not in the file owner class and if the effective group ID or one of the supplementary group IDs of the process matches the group ID associated with the file. [POSIX.1e]

2.2.2.11 information label: The representation of a security attribute of a subject or object that applies to the data contained in that subject or object and is not used for mandatory access control. [POSIX.1e]

2.2.2.12 information label floating: The operation whereby one information label is combined with another information label. The specific algorithm used to define the result of a combination of two labels is implementation defined. [POSIX.1e]

2.2.2.13 MAC label: The representation of a security attribute of a subject or object which represents the sensitivity of the subject or object and is used for mandatory access control decisions. The contents of MAC labels are implementation-defined. [POSIX.1e]

2.2.2.14 mandatory access control (MAC): A means of determining and enforcing access to objects based on an implementation-defined security policy using MAC labels and the use of the implementation-defined dominate operator. The determinations are mandatory in the sense that that are always imposed by the system. [POSIX.1e]

2.2.2.15 minimum ACL: An ACL that contains only the required ACL entries. [POSIX.1e]

2.2.2.16 principle of least privilege: A security design principle that states that a process or program be granted only those privileges necessary to accomplish its legitimate function, and only for the time that such privileges are actually required. [POSIX.1e]

2.2.2.17 required ACL entries: The three ACL entries that must exist in every valid ACL. These entries are exactly one entry each for the owning user, the owning group, and other users not specifically enumerated in the ACL.

2.2.2.18 security: The set of measures defined within a system as necessary to adequately protect the information to be processed by the system. [POSIX.1e]

⇒ **2.3 Built-In Utilities** *Add the following entry to Table 2-3.*

getpcap

⇒ 2.9 Dependencies on Other Standards

⇒ 2.9.1 Features Inherited From POSIX.1

⇒ **2.9.1.4 File Read, Write, and Creation (POSIX.2: line 3395)** *Replace line 3395 in Section 2.9.1.4 with the following:*

(3) If {_POSIX_ACL} is in effect and {_POSIX_ACL_EXTENDED} is in effect for the directory that will contain the new file, the ACL shall be set as described in POSIX.1 {8}, section 23.1.4; otherwise the file permission bits are set to:

⇒ **2.9.1.4 File Read, Write, and Creation (POSIX.2: line 3403)** *Insert the following after line 3403:*

(7) If {_POSIX_CAP} is in effect and {_POSIX_CAP_PRESENT} is in effect for the directory that will contain the new file, the permitted, inheritable and effective capability flags for all capabilities defined in the implementation shall be cleared.

(8) If {_POSIX_INF} is in effect and {_POSIX_INF_PRESENT} is in effect for the directory that will contain the new file, the information label of the file shall be set to an implementation-defined value which should be equivalent to the value returned by the POSIX.1 {8} *inf_default*() function. %

(9) If {_POSIX_MAC} is in effect and {_POSIX_MAC_PRESENT} is in effect for the directory that will contain the new file, the MAC label of the file shall be set to the MAC label of the creating process.

⇒ **2.9.3 Concepts Derived from the Security Standard** *Add this as a new section.*

Some of the standard utilities specify that a utility performs actions equivalent to a POSIX.1 function. In POSIX.1e, if {_POSIX_CAP} is in effect, many functions are associated with specific capability overrides. The behavior of these functions is different between processes whose effective flag is set for the specific capability and processes whose effective flag is clear for the specific capability. Specific utility actions with respect to capabilities are unspecified for utilities in POSIX.2. The concept of user authorization to invoke privileged utility functions is left unspecified in POSIX.1e, and therefore utility enforcement of the authorization mechanism must remain implementation-defined in POSIX.2c.

⇒ **2.13.2 Symbolic Constants for Portability Specifications (POSIX.2: line 4238)** *Insert the following entries in alphabetical order in Table 2-19:*

Table 2-19 - Optional Facility Configuration Values

Name	Description
{POSIX2_ACL}	The system supports the Access Control List Utilities Option (see Section 8).
{POSIX2_CAP}	The system supports the Capability Utilities Option (see Section 9).
{POSIX2_INF}	The system supports the Information Label Utilities Option (see Section 11).
{POSIX2_MAC}	The system supports the Mandatory Access Control Utilities Option (see Section 10).

Section 4: Revisions to Execution Environment Utilities

⇒ **4.13.2 cp — Copy files — Description (POSIX.2: line 2659)** *Change "POSIX.1 {8}" to "POSIX.1 as amended by POSIX.1e {8}" on line 2659 in Section 4.13.2.*

⇒ **4.13.3 cp — Copy files — Description (POSIX.2: line 2708)** *Replace line 2708 in Section 4.13.3 with the following:*

(3) The file permission bits and the S_ISUID and S_ISGID bits.

(4) If {_POSIX_ACL} is defined and {_POSIX_ACL_EXTENDED} is in effect for the destination file, the ACLs. If this fails for any reason, `cp` shall write a diagnostic message to the standard error, do nothing more with the current *source_file* and go on with any remaining files.

⇒ **4.39.2 ls — List directory contents — Description (POSIX.2: line 6024)** *Insert the following sentence after line 6024 in Section 4.39.6.1:*

If {_POSIX_ACL} is defined and {_POSIX_ACL_EXTENDED} is in effect for the file, then the <*optional alternate access method flag*> shall be a plus sign ("+").

⇒ **4.43.2 mv — Move files — Description (POSIX.2: line 7129)** *Insert the following entries after line 7129 in section 4.43.2:*

(6) If {_POSIX_ACL} is defined and {_POSIX_ACL_EXTENDED} is in effect for *dest_file*, then the ACLs associated with the *dest_file* shall reflect the ACLs associated with the *source_file*. If this fails for any reason, `mv` shall write a diagnostic message to the standard error, do nothing more with the current *source_file* and go on with any remaining files.

(7) If {_POSIX_MAC} is defined and {_POSIX_MAC_PRESENT} is in effect for *dest_file*, then the MAC label of *dest_file* shall be set to the MAC label of the invoking process.

(8) If {_POSIX_INF} is defined, and {_POSIX_INF_PRESENT} is in effect for *dest_file* the information label of *dest_file* shall dominate the information label of the source file.

(9) If {_POSIX_CAP} is defined and {_POSIX_CAP_PRESENT} is in effect for *dest_file*, then all capabilities defined by the implementation shall be cleared for *dest_file*.

⇒ **4.48.2 pax — Portable Archive Interchange — Description (POSIX.2: line 7671)** *Insert the following text in line 7671 before the word "access" in section 4.48.2:*

access control lists,

⇒ **4.48.3 pax — Portable Archive Interchange — Description (POSIX.2: line 7774)** *Insert the following after "bits (see 2.2.2.71)," in lines 7774-7775 in section 4.48.3:*

and access control lists,

⇒ **4.48.3 pax — Portable Archive Interchange — Description (POSIX.2: line 7780)** *Insert the following entries after line 7780 in section 4.48.3:*

I Preserve information labels associated with files.

M Preserve mandatory access control labels associated with files.

C Preserve capability state associated with files.

Section 5: POSIX.2— Revisions to User Portability Utilities

⇒ **5.2.2 at — Execute utilities at a later time — Description (POSIX.2: line 87)** *Insert the following after 'mask,' on line 85 in Section 5.2.2:*

> ,the process MAC label (if {_POSIX_MAC} is defined), the process information label (if {_POSIX_INF} is defined), audit ID of the parent job (if {_POSIX_AUD} is defined), the process's capability state (if {_POSIX_CAP} is defined),

⇒ **5.5.2 crontab — Schedule periodic background work — Description (POSIX.2: line 489)** *Insert the following after line 489 in Section 5.5.2:*

The security attributes of the command executed from the *crontab* entry shall be set as follows:

- If {_POSIX_MAC} is defined, a separate *crontab* entry shall be maintained for each MAC label at which the user invokes the `crontab` utility. The MAC label of the environment shall be the MAC label of the process invoking the `crontab` utility.
- If {_POSIX_INF} is defined, the information label of the environment shall be the information label of the process invoking the `crontab` utility.
- If {_POSIX_AUD} is defined, the audit ID of the environment shall be the audit ID of the process invoking the `crontab` utility.
- If {_POSIX_CAP} is defined, the value of the inheritable capability flags in the environment shall be implementation-defined.

Additional implementation-defined restrictions may be imposed when the periodically scheduled job is executed.

Section 8: Access Control Lists

This section describes utilities for the retrieval, modification, and manipulation of access ACLs and default ACLs on specified objects.

Support for the utilities defined in this section is optional but shall be provided by any implementation claiming conformance to the Access Control List Utilities Option. Such an implementation shall provide all of the utilities as described in this section.

8.1 getfacl — Get ACL Information

8.1.1 Synopsis

`getfacl [-d] [`*file...*`]` %

8.1.2 Description

The `getfacl` utility writes discretionary access control information associated with the specified file(s) to standard output. If the `getconf` utility indicates that {_POSIX_ACL_EXTENDED} is not in effect for a *file* then the standard discretionary access permissions are interpreted as an ACL containing only the required ACL entries.

8.1.3 Options

The `getfacl` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

`-d` The operation applies to the default ACL of a directory instead of the access ACL. An error shall be generated if a default ACL cannot be associated with *file*.

8.1.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file whose ACL shall be retrieved. If *file* is not specified, or a *file* is specified as "–", then `getfacl` shall read a list of pathnames, each terminated by one <newline> character, from the standard input. If a pathname read from standard input contains only a <newline> character, the results are unspecified.

8.1.5 External Influences

8.1.5.1 Standard Input

If no *file* operand is specified, or a *file* is specified as a "–", then `getfacl` shall read a list of zero or more pathnames from standard input. Otherwise, standard input shall not be used.

8.1.5.2 Input Files

None.

8.1.5.3 Environment Variables

The following environment variables shall affect the execution of `getfacl`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and standard input).

LC_MESSAGES This variable shall determine the language in which messages should be written.

8.1.5.4 Asynchronous Events

Default.

8.1.6 External Effects

8.1.6.1 Standard Output

The `getfacl` utility writes to standard output a header followed by ACL entries in the form described in 8.1.7. The ACL entries shall be written in the order in which they are evaluated when a discretionary access check is performed. If the –d option is specified and no default ACL is associated with a *file*, then only the header shall be written for that file.

The header shall be written in the following format:

"#file:%s\n#owner:%d\n#group:%d\n",<*filename*>,<*uid*>,<*gid*>

Additional implementation-defined lines starting with a number sign (#) character may be added to the header after the lines specified above.

If more than one ACL is written to standard output, an empty line shall be written to standard output before each header except the first.

8.1.6.2 Standard Error

Used only for diagnostic messages.

8.1.6.3 Output Files

None.

8.1.7 Extended Description

The `getfacl` utility shall write ACL entries in the following form:

<*acl_entry*>
[<*acl_entry*>] ...

Each <*acl_entry*> line shall contain one ACL entry with three required colon-separated fields: an ACL entry tag type, an ACL entry qualifier, and the discretionary access permissions. An implementation may define additional colon-separated fields after the required fields. Comments may be included on any <*acl_entry*> line. If a comment starts at the beginning of a line, then the entire line shall be interpreted as a comment.

The first field contains the ACL entry tag type. This standard defines the following ACL entry tag type keywords, one of which shall appear in the first field:

`user` A `user` ACL entry specifies the access granted to either the file owner or a specified user.

`group` An `group` ACL entry specifies the access granted to either the file owning group or a specified group.

`other` An `other` ACL entry specifies the access granted to any process that does not match any `user`, `group`, or implementation-defined

ACL entries.

`mask` A `mask` ACL entry specifies the maximum access which can be granted by any ACL entry except the `user` entry for the file owner and the `other` entry.

An implementation may define additional ACL entry types.

The second field contains the ACL entry qualifier (referred to in the remainder of this section as qualifier). This standard defines the following qualifiers:

uid This qualifier specifies a user name or a user ID number.

gid This qualifier specifies a group name or a group ID number.

empty This qualifier specifies that no *uid* or *gid* information is to be applied to the ACL entry. An *empty* qualifier shall be represented by an empty string or by white space.

An implementation may define additional qualifiers.

The third field contains the discretionary access permissions. This standard defines the following symbolic discretionary access permissions:

`r` Read access

`w` Write access

`x` Execute/search access

– No access by this ACL entry.

The discretionary access permissions field shall contain exactly one each of the following characters in the following order: `r`, `w`, and `x`. Each of these may be replaced by the "-" character to indicate no access. An implementation may define additional characters following the required characters that represent implementation-defined permissions.

A `user` entry with an *empty* qualifier shall specify the access granted to the file owner. A `user` entry with a *uid* qualifier shall specify the access permissions granted to the user name matching the *uid* value. If the *uid* value does not match a user name, then the ACL entry shall specify the access permissions granted to the user ID matching the numeric *uid* value.

A `group` entry with an *empty* qualifier shall specify the access granted to the file owning group. A `group` entry with a *gid* qualifier shall specify the access permissions granted to the group name matching the *gid* value. If the *gid* value does not match a group name, then the ACL entry shall specify the access permissions granted to the group ID matching the numeric *gid* value.

The `mask` and `other` entries shall contain an *empty* qualifier. An implementation may define additional ACL entry types that use the *empty* qualifier.

A number-sign (#) starts a comment on an <*acl_entry*> line. A comment may start at the beginning of a line, after the required fields and after any implementation-defined, colon-separated fields. The end of the line denotes the end of the comment.

If an ACL entry contains permissions that are not also contained in the `mask` entry, then the output text form for that <*acl_entry*> line shall be displayed as described above followed by a number-sign (#), the string "effective: ", and the effective access permissions for that ACL entry.

White space is permitted in <*acl_entry*> lines as follows: at the start of the line; immediately before and after a ":" separator; immediately before the first number-sign (#) character; at any point after the first number-sign (#) character.

Comments shall have no effect on the discretionary access check of the object with which they are associated. An implementation shall define whether or not comments are stored with an ACL.

If an implementation allows the colon character ":" to be present in an ACL entry qualifier, then that implementation shall provide a method for distinguishing between a colon character as a field separator in an ACL entry definition and a colon character as a component of the ACL entry qualifier value.

8.1.8 Exit Status

The `getfacl` utility shall exit with one of the following values:

0 The ACL for the specified file(s) was successfully retrieved and written to standard output.

>0 An error occurred.

8.1.9 Consequence of Errors

Default.

8.2 setfacl — Set Access Control List

8.2.1 Synopsis

`setfacl` **[**`-bdkn`**] [**`-m` *entries***] [**`-M` *file1***] [**`-x` *entries***] [**`-X` *file2***] [***file...***]**

8.2.2 Description

The `setfacl` utility changes discretionary access control information associated with the specified file(s).

8.2.3 Options

The setfacl utility shall conform to the utility argument syntax guidelines described in 2.10.2.

–b — Remove all entries except the three required base entries.

–d — The operation applies to the default ACL instead of the access ACL. With this option, any *file* arguments must refer to files that may have default ACLs (e.g., directories).

–k — Delete any default ACLs on the specified files. It shall not be considered an error if one or more specified files could, but do not, have a default ACL. An error shall be reported if one or more specified files cannot have a default ACL. The –k option does not require that the –d option be specified. If the –k option is specified, but the –d option is not specified, then all other options apply to the access ACL and not to the default ACL.

–m *entries* — Modify the access or default ACL by adding new entries and updating existing entries with the entries specified in *entries*. The *entries* option argument is a list of comma-separated ACL entries. Each ACL entry shall be in the form described in 8.2.7.1. Permissions in each ACL entry shall be specified by either an absolute value or a relative value. See 8.2.7 for a discussion of how absolute values and relative values are used.

–M *file1* — Modify the access or default ACL by adding new entries and updating existing entries with the entries specified in the pathname *file1*. If *file1* is specified as "–", setfacl will read entries from the standard input.

–n — Do not recalculate the permissions associated with the ACL mask entry.

–x *entries* — Remove the ACL entries specified in *entries* from the access or default ACL of the specified files. The *entries* option argument is a list of comma-separated ACL entries. Each ACL entry shall be in the form described in 8.2.7.1. The permissions field and the preceding colon separator may be omitted from each ACL entry specified in *entries*. If the permissions field is provided, then the value of the permissions field shall be ignored during the processing of the –x option.

–X *file2* — Remove the ACL entries specified in the pathname *file2* from the access or default ACL on the specified files. If *file2* is specified as "–", setfacl will read entries from the standard input. The permissions field and the preceding colon separator may be omitted from each ACL entry specified in *file2*. If the permissions field is provided, then the value of the permissions field shall be ignored during the processing of the –X option.

The –b, –k, –m, –M, –x and –X options shall be evaluated in the order in which they are specified in the invocation of the utility.

8.2.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file on which the specified actions are performed.

8.2.5 External Influences

8.2.5.1 Standard Input

If no *file* operands are specified or if a *file* operand of “–” is specified, `setfacl` shall read the list of pathnames on which to operate from standard input. The format used shall be:

"%s\n", <*file*>

The results are unspecified if a pathname read from standard input contains a <newline> character.

If –M *file1* or –X *file2* are specified and either *file1* or *file2* is “–”, ACL entries are read from standard input as specified in 8.2.5.2.

Standard input shall not be read if a file operand is specified, no *file* operand is “–”, no *file1* option argument is “–”, and no *file2* option argument is “–”.

The results are unspecified if more than one reference is made to standard input by having no *file* operands, a *file* operand of “–”, a *file1* option argument of “–”, and a *file2* option argument of “–”.

8.2.5.2 Input Files

When the –M option is specified, it shall be followed by a *file1* argument. The file specified by the *file1* operand shall contain one or more text form representations of an ACL entry. Each text form representation of an ACL entry shall be in the form described in 8.1.7. ACL entries shall be separated by <newline>. The specified ACL entries shall be added to or updated in the access or default ACL(s) on the specified file(s). The permissions of each ACL entry shall be specified by either an absolute value or a relative value.

When the –X option is specified, it shall be followed by a *file2* argument. The file specified in the *file2* operand shall contain one or more text form representations of an ACL entry. Each text form representation of an ACL entry shall be in the form described in 8.1.7. ACL entries shall be separated by <newline>. The specified ACL entries shall be removed from the access or default ACL(s) on the specified file(s).

8.2.5.3 Environment Variables

The following environment variables shall affect the execution of `setfacl`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

8.2.5.4 Asynchronous Events

Default.

8.2.6 External Effects

8.2.6.1 Standard Output

None.

8.2.6.2 Standard Error

Used only for diagnostic messages.

8.2.6.3 Output Files

None.

8.2.7 Extended Description

In all cases, if the resulting access or default ACL would not be valid, then the utility shall fail for the current file and the access ACL, the default ACL, and the file permission bits shall not be changed. This validity check is not performed until all operations indicated by the specified options have been completed, i.e., at any interim point in the manipulation of the ACL, the internal form of an ACL may be "ill-formed," but it must be valid when the manipulations have been completed.

Two ACL entries shall be considered to match if their tag types are equal and their tag qualifiers are equal.

If the –b option is specified, then all entries other than the three required base entries shall be removed from the ACL. If the ACL contains a `mask` entry, then the permissions associated with the owning `group` entry in the resulting ACL shall be set to only those permissions associated with both the owning `group` entry and the `mask` entry of the current ACL.

If the –m option is specified, the access or default ACL associated with a file operand shall be modified by adding new entries and updating existing entries with each of the ACL entries in the ACL representation specified by *entries*.

If the –M option is specified, the access or default ACL associated with a file operand shall be modified by adding new entries and updating existing entries with each of the ACL entries contained within *file1*.

For both the –m and –M options, if permissions are specified by an absolute value and a matching entry is found, the entire new entry, including permissions, shall replace the current matched entry. If permissions are specified by an absolute value and a matching entry is not found in the ACL, then the entire new entry shall be added to the ACL. If permissions are specified by a relative value and a matching entry is found, the new permissions shall be computed by adding or removing the relative permissions to or from, as appropriate, the permissions in the matching entry. Permissions which are specified to be removed and which are not contained in the permissions of the matching ACL entry shall have no effect on the resulting permissions in the entry. The entire new entry, including the computed permissions, shall replace the current matched entry. If permissions are specified by a relative value and a matching entry is not found in the ACL, then the new entry containing only those permissions specifically granted by the relative value shall be added to the ACL. If no permissions are specified as being added to the entry or if the relative value specifies only the removal of permissions, then the new entry containing no permissions shall be added to the ACL.

For both the –m and –M options, if a `mask` entry is specified, then the permissions of the `mask` entry in the resulting ACL shall be set to the permissions in the specified ACL `mask` entry. If no `mask` entry is specified and the –n option is not specified, then the permissions of the resulting ACL `mask` entry shall be set to the union of the permissions associated with all entries which belong to the file group class in the resulting ACL after all –b, –k, –m, –M, –x, and –X operations have been performed. If no `mask` entry is specified and the –n option is specified, then the permissions of the resulting ACL `mask` entry shall remain unchanged from the existing ACL(s) associated with the *file* operands. If no `mask` entry is specified, the –n option is specified, and no ACL `mask` entry exists in the ACL associated with a file operand, then the `setfacl` utility shall write an error message to standard error and continue with the next file.

If the –x option is specified, those entries in the access or default ACL associated with a file operand which match entries in the ACL representation specified by *entries* shall be removed.

If the –X option is specified, those entries in the access or default ACL associated with a file operand which match ACL entries contained within *file2* shall be removed.

For both the –x and –X options, if a `mask` entry is specified, then the `mask` entry shall be removed from the existing ACL. If no `mask` entry is specified and the –n option is not specified, then the permissions of the resulting ACL `mask` entry shall be set to the union of the permissions associated with all entries which belong to the file group class in the resulting ACL after all –b, –k, –m, –M, –x, and –X operations have been performed. If no `mask` entry is specified and the –n option is specified, then the permissions of the resulting ACL `mask` entry shall remain unchanged in the ACL(s) associated with the *file* operands.

8.2.7.1 ACL Text Format

For both the –m and –x options, the `getfacl` utility shall accept a list of ACL entries in the following form:

<*acl_entry*>[,<*acl_entry*>]...

Each <*acl_entry*> shall contain one ACL entry, as defined in 8.1.7, with two exceptions.

The ACL entry tag type keyword shall appear in the first field in either its full unabbreviated form or its single letter abbreviated form. The abbreviation for `user` is “`u`”, the abbreviation for `group` is “`g`”, the abbreviation for `other` is “`o`”, and the abbreviation for `mask` is “`m`”. An implementation may define additional ACL entry tag type abbreviations.

There are no exceptions for the second field in the short text form for ACLs.

The discretionary access permissions shall appear in the third field. The symbolic string shall contain at most one each of the following characters in any order: `r`, `w`, and `x`; implementations may define additional characters that may appear in any order within the string.

8.2.8 Exit Status

The `setfacl` utility shall exit with one of the following values:

0	Executed successfully and all requested changes were made.
>0	An error occurred.

8.2.9 Consequence of Errors

Default.

Section 9: Capability %

This section describes utilities for the retrieval, modification, and manipulation of the capability state of files, and the retrieval of the capability state for processes.

Three utilities are specified to support capability operations: `getfcap,` `getpcap,` and `setfcap.`

Support for the utilities defined in this section is optional but shall be provided by any implementation claiming conformance to the Capability Utilities Option. Such an implementation shall provide all of the utilities as described in this section.

9.1 getfcap — Get the Capability State of a File

9.1.1 Synopsis

`getfcap` **[** `−m|−M` *flag_spec* **] [** *target...* **]** %

9.1.2 Description

The `getfcap` utility writes the capability state of the specified target files to standard output.

9.1.3 Options

The `getfcap` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−m` Produce output for only those capabilities that have at least one flag set. The default is to produce output for all capabilities defined by the implementation.

`−M` *flag_spec* Produce output only for those capabilities that have at least one of the flags specified in *flag_spec* set. The default is to produce output for all capabilities defined by the implementation. *flag_spec* contains one or more character(s), each of which represents a capability flag defined in the implementation:

WITHDRAWN DRAFT. All Rights Reserved by IEEE.
Preliminary—Subject to Revision.

- `e` Specifies the effective capability flag.
- `i` Specifies the inheritable capability flag.
- `p` Specifies the permitted capability flag.

Future revisions of this standard may use other lowercase letters in the portable filename character set in *flag_spec*. Uppercase letters in the portable filename character set are reserved for implementations to refer to implementation-defined capability flags.

9.1.4 Operands

The following operand shall be supported by the implementation:

target *Target* represents the pathname(s) of the file(s) whose capability state shall be displayed. If no *target* is specified or if a target operand is "–", `getfcap` shall read a list of zero or more pathnames, each terminated by one <newline> character, from standard input. If a pathname read from standard input contains only a <newline> character, the results are unspecified. This list of pathnames read from standard input shall be terminated by end-of-file (`EOF`).

9.1.5 External Influences

9.1.5.1 Standard Input

The standard input shall be used only if no *target* operands are specified, or if a *target* operand is "–". If the use of standard input is specified, `getfcap` shall read a list of zero or more pathnames from standard input.

9.1.5.2 Input Files

None.

9.1.5.3 Environment Variables

The following environment variables shall affect the execution of `getfcap`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single-versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

9.1.5.4 Asynchronous Events

Default.

9.1.6 External Effects

9.1.6.1 Standard Output

A textual representation of the capability state of the target file(s) specified will be written to the standard output. If multiple targets are specified, the output for each target will be preceded by the identifier for that target. The format shall be:

"%s:\n", *<target>*

The state of the capabilities for each target shall be written according to the text form representation specified.

9.1.6.2 Standard Error

Used only for diagnostic messages.

9.1.6.3 Output Files

None.

9.1.7 Extended Description

None.

9.1.8 Exit Status

The `getfcap` utility shall exit with one of the following values:

0 The capability state of the specified target was successfully reported.

>0 An error occurred.

9.1.9 Consequence of Errors

The file pathname that was being examined when the error occurred shall be written to the error output file along with a brief error message. Processing of that target shall immediately be terminated, and the command shall continue with the next target, if specified.

9.2 getpcap — Get the Capability State of a Process

9.2.1 Synopsis

`getpcap` **[** `−m|−M` *flag_spec* **]** %

9.2.2 Description

The `getpcap` utility writes the capability state of the invoking process to standard output.

9.2.3 Options

The `getpcap` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−m` Produce output for only those capabilities that have at least one flag set, and for only those capability attributes that are set. The default is to produce output for all capabilities defined by the implementation.

`−M` *flag_spec* Produce output only for those capabilities that have at least one of the flags specified in *flag_spec* set. The default is to produce output for all capabilities defined by the implementation. *flag_spec* contains one or more character(s), each of which represents a capability flag defined in the implementation:

`e` Specifies the effective capability flag.

`i` Specifies the inheritable capability flag.

p Specifies the permitted capability flag.

Future revisions of this standard may use other lowercase letters in the portable filename character set in *flag_spec*. Uppercase letters in the portable filename character set are reserved for implementations to refer to implementation-defined capability flags.

9.2.4 Operands

None.

9.2.5 External Influences

9.2.5.1 Standard Input

None.

9.2.5.2 Input Files

None.

9.2.5.3 Environment Variables

The following environment variables shall affect the execution of getpcap:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

9.2.5.4 Asynchronous Events

Default.

9.2.6 External Effects

9.2.6.1 Standard Output

The `getpcap` utility writes to standard output the textual representation of the capability state of the invoking process in the text form specified in 9.1. The following format shall be used:

"%s:\n", *<process_cap>*

9.2.6.2 Standard Error

Used only for diagnostic messages.

9.2.6.3 Output Files

None.

9.2.7 Extended Description

None.

9.2.8 Exit Status

The `getpcap` utility shall exit with one of the following values:

0 The capability state of the specified target(s) was(were) successfully reported.

>0 An error occurred.

9.2.9 Consequence of Errors

None.

9.3 setfcap — Set Capability State of a File

9.3.1 Synopsis

```
setfcap −e state [ −e state |−f state_file ] ... [target... ]          %
setfcap −f state_file [ −e state |−f state_file ] ... [target... ]     %
setfcap state [target... ]
```

9.3.2 Description

The `setfcap` utility changes the capability state associated with the specified file(s). The state to be set is defined by the *–e* option, the *–f* option, or the *state* operand. If no *–e* options and no *-f* options are specified, the last form is assumed. The *state* operand's value consists of one or more capabilities as defined by the text form representation specified in 9.1. If a complete capability state is specified, that capability state completely replaces any existing capability state on the file. If only a partial capability state is specified, the old capability state will be a modified to reflect the specification.

9.3.3 Options

The `setfcap` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

`–e` *state* Specify a partial or complete capability state, consisting of one or more capability specifications, to be assigned to each of the target(s). Capability specifications in *state* shall be separated by a `<comma>` or `<newline>`. A null capability specification can be specified by two adjacent `<newline>`s or `<comma>`s in *state*. Null capability specifications shall be ignored. The capability specification is described by the text form representation specified in 9.1.

`–f` *state_file* Read one or more capability specifications that represent a partial or complete capability state from the file named by the pathname *state_file*. Capability specifications in *state_file* shall be terminated by a `<newline>`. A null capability specification can be specified by an empty line in *state_file*. Null capability specifications shall be ignored. If *state_file* is specified as a dash "–", `setfcap` shall read capability specifications from the standard input file.

Multiple `–e` and `–f` options shall be accepted by the `setfcap` utility, in which case the capability specifications shall be processed in the order they are specified.

9.3.4 Operands

The following operands shall be defined by the implementation:

state *State* represents the partial or complete capability state to be assigned. Capability specifications in *state* shall be separated by a `<comma>` or `<newline>`. A null capability specification can be specified by two adjacent `<newline>`s or `<comma>`s in *state*. Null capability specifications shall be ignored. The capability specification is described by the text form representation specified in 9.1.

target *Target* represents the name(s) of the file(s) whose capability state is (are) to be modified. If no *target* is specified, or is "–", and no *state_file* argument is specified as a dash "–", `setfcap` shall read a list of pathnames, separated by one or more white space characters, from the standard input file, terminated by end-of-file (`EOF`).

9.3.5 External Influences

9.3.5.1 Standard Input

The `setfcap` utility shall read capability state information from standard input if a *state_file* option argument is "–" (see 9.4.3) and shall read pathnames from standard input if a *target* operand is "–" or if no *target* operands are specified (see 9.4.4). Otherwise, standard input shall not be read. The results are unspecified if more than one reference is made to standard input by having "–" as a statefile option argument, having no operands, or by having "–" as a target operand.

9.3.5.2 Input Files

The *state_file* option argument shall be interpreted as the pathname of a file that contains a set of correct external representations of capability states, as described by the text form representation specified.

9.3.5.3 Environment Variables

The following environment variables shall affect the execution of `setfcap`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

9.3.5.4 Asynchronous Events

Default.

9.3.6 External Effects

9.3.6.1 Standard Output

None.

9.3.6.2 Standard Error

Used only for diagnostic messages.

9.3.6.3 Output Files

None.

9.3.7 Extended Description

None.

9.3.8 Exit Status

The `setfcap` utility shall exit with one of the following values:

0 The specified capability state changes were successfully made.

>0 An error occurred.

9.3.9 Consequences of Errors

In the event of an error, the capability state of the target that caused the error shall not be modified, and `setfcap` shall continue on to the next file. Each diagnostic message indicating that the capability state of a target could not be changed shall include the name of the target.

Section 10: Mandatory Access Control %

This section describes the utilities that shall be implemented on all systems that claim conformance to the Mandatory Access Control Utilities Option.

Support for the utilities defined in this section is optional but shall be provided by any implementation claiming conformance to the Mandatory Access Control Utilities Option. Such an implementation shall provide all of the utilities as described in this section.

Three utilities are specified to support mandatory access control. The `getfmac` utility provides the means for a user to display the MAC label of a file. The `getpmac` utility provides the means for a user to display the MAC label of the current process. The `setfmac` utility provides the means to set the MAC label of a file.

10.1 getfmac — Get the MAC Label of a File

10.1.1 Synopsis

`getfmac [`*file...*`]`

10.1.2 Description

The `getfmac` utility writes to standard output the text form of MAC labels. For each *file* operand, the `getfmac` utility shall perform the equivalent to the POSIX.1e *mac_get_file*() and *mac_to_text*() functions, and write the returned text string to standard output.

The `getfmac` utility requires mandatory read access to each file for which the label has been requested.

10.1.3 Options

None.

WITHDRAWN DRAFT. All Rights Reserved by IEEE.
Preliminary—Subject to Revision.

10.1.4 Operands

The following operand shall be supported by the implementation:

file The pathname of a file whose MAC label is to be written. If *file* is not specified, or a *file* is specified as a "–", then `getfmac` shall read a list of pathnames, each terminated by one <newline> character, from the standard input.

10.1.5 External Influences

10.1.5.1 Standard Input

If no *file* operand is specified, or a *file* is specified as a "–", then `getfmac` will accept a list of zero or more pathnames, each terminated by one <newline> character, from the standard input.

10.1.5.2 Input Files

None.

10.1.5.3 Environment Variables

The following environment variables shall affect the execution of `getfmac`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and standard input).

LC_MESSAGES This variable shall determine the language in which messages should be written.

10.1.5.4 Asynchronous Events

Default.

10.1.6 External Effects

10.1.6.1 Standard Output

The following format shall be used for each file processed:

"%s:\t%s\n",<*file name*>,<*file_MAC_label*>

The output format of the <*file_MAC_label*> shall be suitable for re-input as the *label* operand to the `setfmac` utility on the same system.

10.1.6.2 Standard Error

Used only for diagnostic messages.

10.1.6.3 Output Files

None.

10.1.7 Extended Description

None.

10.1.8 Exit Status

The `getfmac` utility shall exit with one of the following values:

0	The utility executed successfully.
>0	An error occurred.

10.1.9 Consequence of Errors

Default.

10.2 getpmac — Get Text Form Of Current Process's MAC Label

10.2.1 Synopsis

```
getpmac
```

10.2.2 Description

The `getpmac` utility writes the text form of the MAC label of the current process.

10.2.3 Options

None.

10.2.4 Operands

None.

10.2.5 External Influences

10.2.5.1 Standard Input

None.

10.2.5.2 Input Files

None.

10.2.5.3 Environment Variables

The following environment variables shall affect the execution of `getpmac`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6. %

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.–

LC_MESSAGES This variable shall determine the language in which messages should be written.

10.2.5.4 Asynchronous Events

Default.

10.2.6 External Effects

10.2.6.1 Standard Output

The following output format shall be used;

"%s\n",*<process_MAC_label>*

The output format of the *<process_MAC_label>* shall be suitable for re-input as the *label* operand to the `setfmac` utility on the same system.

10.2.6.2 Standard Error

Used only for diagnostic messages.

10.2.6.3 Output Files

None.

10.2.7 Extended Description

None.

10.2.8 Exit Status

The `getpmac` utility shall exit with one of the following values:

0 The utility executed successfully.

>0 An error occurred.

10.2.9 Consequence of Errors

Default.

10.3 setfmac — Set the MAC Label of a File

10.3.1 Synopsis

`setfmac` *label* **[***file...***]**

10.3.2 Description

The `setfmac` utility changes the MAC label of each specified file to the label specified by *label*. For each *file* operand, the `setfmac` utility shall perform the equivalent to the POSIX.1e *mac_from_text*() and *mac_set_file*() functions.

10.3.3 Options

None.

10.3.4 Operands

The following operands shall be supported by the implementation:

label The textual representation of the MAC label specified.

file The pathname of a file whose MAC label is to be changed. If *file* is not specified, or a *file* is specified as a "–", then `setfmac` shall read a list of pathnames, each terminated by one <newline> character, from the standard input. If a pathname read from standard input contains only a <newline> character, the results are unspecified.

10.3.5 External Influences

10.3.5.1 Standard Input

If no *file* operand is specified, or a *file* is specified as a "–", then `setfmac` will accept a list of one or more pathnames, each terminated by one <newline> character, from the standard input. The results are unspecified if "–" is specified as a file operand more than once.

10.3.5.2 Input Files

None.

10.3.5.3 Environment Variables

The following environment variables shall affect the execution of `setfmac`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

10.3.5.4 Asynchronous Events

Default.

10.3.6 External Effects

10.3.6.1 Standard Output

None.

10.3.6.2 Standard Error

Used only for diagnostic messages.

10.3.6.3 Output Files

None.

10.3.7 Extended Description

None.

10.3.8 Exit Status

The `setfmac` utility shall exit with one of the following values:

0 The utility executed successfully.

>0 An error occurred.

10.3.9 Consequence of Errors

Default.

Section 11: Information Labeling %

This section describes utilities for the retrieval and manipulation of information labels on specified objects.

Support for the utilities defined in this section is optional but shall be provided by any implementation claiming conformance to the Information Label Option. Such an implementation shall provided all of the utilities as described in this section.

Three utilities are specified to support information labeling. The `getfinf` utility provides the means for a user to display the information label of a file. The `getpinf` utility provides the means for a user to display the information label of the current process. The `setfinf` utility provides the means for a user to set the information label of a file.

11.1 getfinf — Get File Information Label

11.1.1 Synopsis

`getfinf [`*file...*`]`

11.1.2 Description

The `getfinf` utility writes to standard output the text form of information labels. For each *file* operand, the `getfinf` utility shall perform the equivalent to the POSIX.1e *inf_get_file()* and *inf_to_text()* functions, and write the returned text string to standard output.

11.1.3 Options

None.

11.1.4 Operands

The following operand shall be supported by the implementation:

file The pathname of a file whose information label is to be written.

11.1.5 External Influences

11.1.5.1 Standard Input

If no *file* operand is specified, or a *file* is specified as a "–", then `getfinf` shall read a list of zero or more pathnames from standard input. Otherwise standard input shall not be read. The results are unspecified if "–" is specified as a file operand more than once.

11.1.5.2 Input Files

None.

11.1.5.3 Environment Variables

The following environment variables shall affect the execution of `getfinf`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and standard input).

LC_MESSAGES This variable shall determine the language in which messages should be written.

11.1.5.4 Asynchronous Events

Default.

11.1.6 External Effects

11.1.6.1 Standard Output

The following format shall be used for each *file* operand specified if multiple operands are specified, or if the *file* operand is specified as “–”.

```
"%s:\t%s\n", <file_name>, <file_information_label>
```

The output format of the label shall be suitable for re-input as the *inflabel* operand to the `setfinf` utility on the same system.

11.1.6.2 Standard Error

Used only for diagnostic messages.

11.1.6.3 Output Files

None.

11.1.7 Extended Description

None.

11.1.8 Exit Status

The `getfinf` utility shall exit with one of the following values:

0 The information labels associated with all specified files were successfully reported.

>0 An error occurred.

11.1.9 Consequences of Errors

Default.

11.2 getpinf — Get Process Information Label

11.2.1 Synopsis

```
getpinf
```

11.2.2 Description

The `getpinf` utility writes the information label associated with the current process to standard output. Note that some floating policies may cause this label to differ from that of the invoking process (e.g., command interpreter).

11.2.3 Options

None.

11.2.4 Operands

None.

11.2.5 External Influences

11.2.5.1 Standard Input

None.

11.2.5.2 Input Files

None.

11.2.5.3 Environment Variables

The following environment variables shall affect the execution of `getpinf`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.–

LC_MESSAGES This variable shall determine the language in which messages should be written.

11.2.5.4 Asynchronous Events

Default.

11.2.6 External Effects

11.2.6.1 Standard Output

The following format shall be used:

"%s\n", <*process_information_label*>

The output format of the label shall be suitable for re-input as the *inflabel* operand to the `setfinf` utility on the same system. +

11.2.6.2 Standard Error

Used only for diagnostic messages.

11.2.6.3 Output Files

None.

11.2.7 Extended Description

None.

11.2.8 Exit Status

The `getpinf` utility shall exit with one of the following values:

0 The information label associated with the invoking process was successfully reported.

> 0 An error occurred.

11.2.9 Consequences of Errors

Default.

11.3 setfinf — Change File Information Label

11.3.1 Synopsis

`setfinf` *inflabel* `[`*file...*`]`

11.3.2 Description

The `setfinf` utility sets the information label associated with each of the specified files to the specified information label.

11.3.3 Options

None.

11.3.4 Operands

The following operands shall be supported by the implementation:

inflabel The new information label to be associated with each of the specified files.

file The pathname of a file whose information label is to be changed. –

11.3.5 External Influences

11.3.5.1 Standard Input

If no *file* operand is specified, or a *file* is specified as a "–", then `setfinf` shall read a list of zero or more pathnames from standard input. Otherwise standard input shall not be read.

11.3.5.2 Input Files

None.

11.3.5.3 Environment Variables

The following environment variables shall affect the execution of `setfinf`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for this interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and standard input).

LC_MESSAGES This variable shall determine the language in which messages should be written.

11.3.5.4 Asynchronous Events

Default.

11.3.6 External Effects

11.3.6.1 Standard Output

None.

11.3.6.2 Standard Error

Used only for diagnostic messages.

11.3.6.3 Output Files

None.

11.3.7 Extended Description

None.

11.3.8 Exit Status

The `setfinf` utility shall exit with one of the following values:

0 The information labels associated with all specified files were successfully changed.

>0 An error occurred.

11.3.9 Consequences of Errors

Default.

Annex E
(informative)

Revisions to the General Section

⇒ **E.2.9.3 Concepts Derived from the Security Standard Rationale** *Add this as a new section.*

This subclause was introduced to describe the relationship between capabilities defined in POSIX.2c and the power or trust associated with a user account, commonly called an authorization in security literature. The POSIX.1 and POSIX.2 standards purposely do not assume the traditional superuser model of trust (effective or real user ID 0), or any other model. Rather, the phrase “appropriate privilege” is included in POSIX.1 to allow for traditional POSIX implementations and trusted system implementations that can support POSIX.1 conforming applications.

The enforcement of power by trusted applications can be based on the identity of the invoking user or on the presence of a trusted program in the process chain that preceded the execution of the utility in the current process. The inheritance of trust, or the indication of a previous trusted process image, through the process chain is the motivation behind the inheritable capabilities flag in the POSIX.2c standard. The state of the user’s initial process inheritable flags at user authentication time is unspecified by POSIX.2c because there is no concept of a user account or a user authentication profile that would normally contain this information. Similarly, the manner in which an administrator or security officer assigns trust or power to user accounts is similarly unspecified by POSIX.2c.

Therefore, the constraints imposed by the utilities defined in POSIX.2 must be specified as behaving differently given “appropriate privilege” or “appropriate authorization.” Until the mechanisms for enforcement are specified by POSIX.2c, this concept must remain undefined.

The two major goals in determining changes to POSIX.2 were to define minimal changes, and to avoid requiring that particular utilities be added to the TCB.

The main reason for avoiding changes to POSIX.1 and POSIX.2 is the lack of consensus of existing practice. For example, some implementations add options to the `ls` utility to display security attributes, while others define new utilities. Even those which add options are inconsistent in the options and the formats used. Additionally, there is no strong justification for adding options to some utilities. For example, options to test to compare MAC labels may be desirable, but it is not yet clear what tests are useful.

Putting particular utilities in the TCB causes a ripple effect, possibly forcing utilities like the shell into the TCB. This was considered undesirable. In order to avoid putting utilities into the TCB, the standard does not specify capabilities required by or used by particular utilities.

For example, the standard does *not* require that the chown utility use the CAP_CHOWN capability if the invoking process includes that capability in its inheritable set, nor does it require that the file containing the chown utility include that capability in its permitted set. Thus, inclusion of the CAP_CHOWN capability in the inheritable set may or may not cause the chown utility to use the capability. Another example is the ps utility, where the standard does not specify what capabilities may be required in order to get information about processes, nor the capability enforced. Neither does the standard specify whether any policy or capabilities are enforced by the ps utility (as in some historical implementations which use */dev/kmem*), or by the underlying system (as in other implementations which use */dev/proc* or similar mechanisms).

Thus, rather than requiring specific capabilites, the standard makes the capabilities required by particular utilities implementation-defined. By not defining capabilities, the standard leaves open utility based authentication using mechanisms outside this standard.

⇒ **E.2.13.2 Symbolic Constants for Portability Specifications (POSIX.2: line 2875)** *Insert the following after line 2875:*

POSIX2_ACL See the rationale in E.8

POSIX2_CAP See the rationale in E.9

POSIX2_INF See the rationale in E.11

POSIX2_MAC See the rationale in E.10

⇒ **E.4.6 chgrp — Change file group ownership** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The chgrp utility refers to the chown function in POSIX.1 for its functionality. While this standard specifies the capabilities required for the function, it does not specify the capabilities requires for the utility.

⇒ **E.4.7 chmod — Change file mode** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The chmod utility does *not* refer to the chmod function in POSIX.1 for the exclusive definition of its functionality, but rather specifies "appropriate privilege." This standard does not define appropriate privilege for utilities.

⇒ **E.4.8 chown — Change file ownership** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The chown utility refers to the chown function in POSIX.1 for its functionality. While this standard specifies the capabilities required for the function, it does not specify the capabilities requires for the utility.

⇒ **E.4.13 cp — Copy files** *Rationale for changes to this section in POSIX.2 is provided below:*

The definition of cp calls for duplicating file permission bits when a new file is created, and for propagating certain characteristics of files when the "–p" option is given. ACLs are included in these propagated characteristics. ACLs are not copied in absence of the "–p" option to maintain compatibility in the cases where the ACL can not be copied.

Note that in the absence of the "–p" option, the cp utility already specifies that new files are created using open(), specifying the file permission bits of the source file in the *mode* argument, and POSIX.1e specifies the impact of default ACLs on open(). The result being that if there is a default ACL on the destination directory, the resulting ACL on the destination file will be the default ACL modified by the permission bits of the source file. This effectively will limit access to the newly created file to the minimum of accesses specified in the default ACL and the source file permissions. If the destination directory does not have a default ACL, then the permission bits of the newly created file will be the source file permission bits as modified by the umask.

Experience with historical operating systems has shown that it is important to be able to specify whether the old ACL is copied (as when the –p option is specified), or whether to apply the normal creation defaults (when the –p option is not specified). While this standard does not require any particular option, implementors are advised to add specific options to copy the old ACL without copying the other attributes brought along when the "–p" option is used.

No specific feature is provided for copying MAC labels, information labels, or capabilities when the "-p" option is provided. Such a feature would require use of appropriate privilege, which this standard avoids wherever possible.

It would appear feasible to specify the information label of the copied file. However, that label may depend on the label of the invoking process and the files being copied. Additionally, it may depend on the order in which the operations are performed. For example, a request to copy several files could cause the information label of the last file copied to float up to include all of the previous files. Such a definition would be too complex to be of any use.

The only statement that is clearly true is that the information label of the destination file will dominate the information label of the invoking process, the information label of the source file, and the information label of the destination file (if it already existed).

⇒ **E.4.16 dd — Convert and copy a file** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The `dd` utility is a sophisticated copy tool frequently used for copying disks and tapes. However, because it relies on the lower level primitives (such as the `open` function in POSIX.1) no changes are required.

⇒ **E.4.24 find — Find files** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

Some consideration was given to adding options to `find` to locate files based on MAC labels, information labels, capabilties, and ACLs. However, there was no overwhelming evidence that such options are necessary. Furthermore, they can be emulated using existing features of the `find` utility. For example, to find a file based on the presence of a particular user in the file's ACL, use the following statement:

```
find directory -exec checkacl {} username \;
```

where *directory* is the directory being searched, *username* is the user being searched for, and checkacl is the following shell script:

```
getfacl $1 | grep -s $2 >/dev/null
if [ $? -eq 0 ]
then
   printf "%s0$1"
fi
```

Similar scripts can be written to find files based on other attributes.

⇒ **E.4.26 getconf — Get configuration values** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The description of the `getconf` utility does not list the configuration parameters, but refers to the appropriate tables in POSIX.1. Because POSIX.1e is updating the POSIX.1 tables, the additional configuration parameters are included in the `getconf` utility by reference.

⇒ **E.4.38 lp — Send files to a printer** *Rationale for why no changes were made to this section in POSIX.2 is provided below:*

The ballot resolution group came to the conclusion that labeling of print output is a policy issue, not an interface issue, and therefore the standard should not require `lp` to print MAC or information labels.

There are several issues associated with `lp`: human readable label format, what label(s) to print, suppression of human readable labels, and rejection of spool requests.

The standard does not address the format of human readable labels because it is not an interface issue. Further, there is no agreement on what is contained in a MAC or information label, so any discussion of the human readable form is pointless.

It is arguable whether the MAC label on printed output should be the MAC label of the file being printed or the MAC label of the process making the print request.

There is some sentiment that the information label of a file and/or process making a print request should also be printed on each page. The standards committee did not see a strong demand for this facility, and hence it is not included in the standard.

Suppression of human readable labels can be considered an interface issue, but it is not *required* by the TCSEC. Rather, it is *allowed* as an exception, providing that auditing is performed. While many implementations will provide this exception, there was no consensus that it is required as an option, especially because the POSIX.2 standard explicitly states that the format of any output is implementation defined.

In a secure system, `lp` may reject spool requests based on criteria such as the physical location of the printer, security level of the requesting user, or time of day. The standard provides a general allowance for any unspecified policy for rejecting or cancelling print requests. The statement "..if such a device is not available to the application..." allows a conforming system to refuse the request.

⇒ **E.4.39 ls — List directory contents** *Rationale for changes to this section in POSIX.2 is provided below:*

There are two issues with regard to `ls`: handling of the alternate access method flag, and displaying additional file information.

The access method flag is used to identify ACLs only. Consideration was given to having it indicate MAC, information labels, and capabilities as well. However, on a system with MAC, all files will have a MAC label. Similarly, on a system with information labels, all files will have information labels. For these cases, the indicator would always be on, so it would provide no additional information. Privileges do not participate in access control decisions, so it was decided that they should not be indicated by an access method flag.

The issue of extending `ls` to display additional information is difficult. The utility `ls` already has a myriad of options, and many more would be needed for security information. Also, the existing paradigm is one line per file. Given the amount of security information (MAC label, information label, ACL, capabilities) which can be on a file, restricting output to a single line is impractical. Thus, no additional options are provided, but rather new utilities are added to display the security relevant data.

⇒ **E.4.43 mv — Move files** *Rationale for changes to this section in POSIX.2 is provided below:*

When a file is being renamed within a file system, the POSIX.2 standard specifies that an error is generated if the renaming fails. This is adequate from the perspective of MAC and ACL. When a file is copied (as a result of a movement to a different file system), the POSIX.2 standard specifies the file characteristics which are copied with the file. Requiring that the file MAC label, information label, and capabilites be retained would require a discussion of appropriate privilege, which this standard avoids.

It would appear that the information label of the destination file could be specified. However, the destination information label depends on whether the destination file is on the same file system as the source file (in which case the operation is a rename, most likely without any change in the information label of the file), or on a separate file system (in which case the operation is a copy, and the information label of the destination file is based on the information label of the process which invoked `mv` as well as the information label of the source file). Hence, this standard does not specify the information label of the destination file.

⇒ **E.4.44 nohup — Invoke a utility immune to hangups** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The `nohup` utility refers to specific permission bits when creating an output file. However, because the bits are referred to as modifications to 2.9.1.4, the other changes to 2.9.1.4 (to specify the file MAC label, etc.) are adequate.

⇒ **E.4.48 pax — Portable archive interchange** *Rationale for changes to this section in POSIX.2 is provided below:*

There are three classes of changes to the `pax` utility: changes to the user interface, changes to the backup format, and use of appropriate privilege.

Extensions to the interface to restore security attributes are provided as part of this standard. Capital letters were selected to avoid conflicts with other specification characters. Note that using one or more of these specification characters may not cause restoration of the corresponding security attributes, because the `pax` utility may still require appropriate privilege. Rather than defining a separate specification character for access control lists, restoration of ACLs is included with file permission bits.

The `pax` utility specifies the interface for creating a backup. Definition of the backup format is outside the scope of POSIX.2, and hence changes to the backup format are outside the scope of this standard. Note that the two backup formats referenced in POSIX.2 (tar and cpio) are not extensible to add security attributes. Hence, the capability to restore security attributes is only present if an implementation dependent backup format is used.

As with other utilities, this utility calls for appropriate privilege, which is not further specified.

⇒ **E.4.53 rm — Remove directory entries** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The `rm` utility refers to file permissions, but without specifying permission bits. Additionally, it allows for arbitrary failure of the directory entry removal. Thus, the definition of the utility is general enough that no change is required.

⇒ **E.4.59 stty — Set the options for a terminal** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The `stty` utility exists only to provide access to the General Terminal Interface. Thus, implementations should add restrictions to this utility consistent with the restrictions placed on the GTI interfaces as noted previously.

⇒ **E.4.62 test — Evaluate expression** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

Some consideration was given to adding options to `test` to compare MAC labels, test for presence of ACLs, etc. However, there was no overwhelming evidence that such options are necessary.

⇒ **E.5.2 at — Execute utilities at a later time** *Rationale for changes to this section in POSIX.2 is provided below:*

Particular implementations may wish to restrict use of this utility, or to require additional authorization checks when the job is actually run. The form of any such restrictions is left implementation defined.

For example, the system could reauthorize the user before the job is executed to verify that the user is still authorized to run at the MAC label and with the capabilties in use at the time the job was queued.

⇒ **E.5.3 batch** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

No changes are required for this utility, because `batch` is defined in terms of `at`.

⇒ **E.5.5 crontab — Schedule periodic background work** *Rationale for changes to this section in POSIX.2 is provided below:*

Some implementations may wish to restrict use of this utility, or to require additional authorization checks when the job is actually run. The form of any such restrictions is left implementation defined.

For example, the system could reauthorize the user before the job is executed to verify that the user is still authorized to run at the MAC label and with the capabilities in use at the time the job was queued.

⇒ **E.5.17 mesg — Permit or deny messages** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The POSIX.2 definition of this utility specifies “appropriate privilege.” This standard does not define what capability is required.

⇒ **E.5.19 newgrp — Change to a new group** *Rationale for changes to this section in POSIX.2 is provided below:*

When *newgrp* changes the group identification, it is important to retain the MAC label, information labels, and inherited capability flags along with the remainder of the process environment.

On traditional implementations, the program requires capabilities in order to change the group ID. Some implementations may require that the invoking process also have capabilities before the utility is executed.

This utility may prompt the user for a password. This is in some sense a form of identification. However, it is only changing the group ID for a user who has already been identified. Hence, this standard does not require any additional restrictions.

⇒ **E.5.20 nice — Invoke a utility with an altered system scheduling priority** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The POSIX.2a definition of this utility specifies "appropriate privilege." This standard does not define what privilege is required.

⇒ **E.5.23 ps — Report process status** *Rationale for the lack of changes to this section in POSIX.2 is provided below:*

The POSIX.2 definition of this utility specifies "appropriate privilege." This standard does not define what privilege is required. For example, a privilege may be required to see processes belonging to a different user or operating at MAC labels other than the MAC label of the invoking process.

E.8 Access Control Lists

E.8.1 User-Level Utilities

Command line interfaces, i.e., utilities, are provided to examine and manipulate ACL entries. There were several major decisions with the utility interfaces. The following subsections explain the rationale for these decisions.

E.8.1.1 Separate Utilities

The functionality specified in the `getfacl` utility could be added to the `ls` utility. However, the `ls` interface is already sufficiently complex and adding an ACL display capability to `ls` would simply further complicate an overly complex interface.

As an alternative, a single utility interface could be provided which would include all of the optional and non-optional utility interfaces specified in this standard. Separate `getfacl` and `setfacl` utilities were specified in order to provide a more modular solution.

E.8.1.2 Utility Names

The names of the `getfacl` and `setfacl` utilities were chosen to be as descriptive as possible of the operations performed by the utilities. The names were also chosen to be consistent with the equivalent utilities in the other sections of the standard.

E.8.1.3 Ease of Use

ACL entries are manipulated by specification of a new external representation or by specification of changes to the existing external representation. The external representation of an ACL entry is not trivial. One of the goals of the working group is to encourage the use of ACLs. This goal is accomplished by making design decisions that are biased towards ease of use. In order to make the utility interface easier to use, the output of the `getfacl` utility for a single file can be used as input to the `setfacl` utility. The `getfacl` utility can be used to list the ACLs of multiple files. However, the resulting output could not be used directly as input to the `setfacl` utility because the `getfacl` output would contain multiple entries for the file owner, file group, and other. An attempt to use this as input to `getfacl` would result in an error because the resulting ACL would not be valid as defined by the acl_valid() function.

The ACL **mask** entry adds significant complexity to the `getfacl` and `setfacl` utility interfaces. This complexity is especially obvious in the `setfacl` utility. In keeping with the goal to provide interfaces which are relatively easy to use, the `setfacl` utility provides a basic set of options to manipulate ACLs including the **mask** entry. The function of automatically generating (or recalculating) the permissions for the **mask** entry was chosen as the default operation for `setfacl` in order to allow most users to manipulate ACLs without requiring direct, conscious manipulation of the mask. For those cases where the mask has specifically been altered to limit the permissions granted by additional entries, the –n option is provided to allow users to manipulate the ACL without affecting the mask. These few operations for the mask should provide the basic capability of manipulating ACLs in most environments. Certainly, additional options may be desirable, e.g. an option to recalculate the mask but remove "extra" permissions that might be granted to other entries by the mask recalculation. Such options were included in a previous draft but were removed due to the overwhelming complexity which they added to the interface. This includes the following:

(1) Default operation to recalculate the mask value but issue warning messages if any ACL entries might inadvertently grant additional access based on the recalculation of the mask.

(2) A –p option to remove any permissions from ACL entries which are limited by the mask value.

(3) A –c option to always recalculate the mask regardless of the effect on the effective permissions granted by ACL entries.

The intent of this interface is to provide a basic set of utilities for manipulating ACLs. Implementations may certainly extend the utility interfaces with these or other options.

It is expected that most implementations will provide more sophisticated ACL editors to improve ease of use. The working group considered specifying this editor but concluded that such an editor would primarily be a screen oriented user interface and should not be required of conforming implementations.

E.8.1.4 Utility Options

Picking utility options is never an easy task. Options and option characters were selected which were, at least, moderately meaningful while maintaining consistency with the use of option characters by other utilities in existing implementations. The rationale for the inclusion of each of the utility options, and the selection of the option characters, for the `setfacl` utility are:

–b This option provides a simple method to reset an ACL to the three base entries (the owner, owning group, and other). This operation could be accomplished by reading the current ACL associated with the file, removing all extended ACL entries, and then updating the ACL with the result. However, resetting a file's ACL to the three base entries appears to be a sufficiently significant and frequent operation as to justify an option to quickly perform the operation. The –b option was chosen to indicate the **base** ACL entries. The owning **group** entry is reset to the intersection of the owning **group** entry and the **mask** entry by the –b option in order to prevent an inadvertent increase in the effective permissions of the owning group when removing the **mask** entry.

–d This option indicates that the requested operations are to be performed on the directory's default ACL instead of the access ACL. This could be implemented with another utility, e.g. a `setdefacl` utility; however, this would result in a second utility with exactly the same options as `setfacl`. Since default and access ACL are manipulated in exactly the same manner and with the same entry validations, a single utility with an option to select the type of ACL is moderately simpler.

–k The –k option entirely removes a default ACL from a directory. This option is necessary, in addition to the –b option, because the –b option only removes the extended entries and leaves the 3 base entries. The –k will completely remove the default ACL from the directory. This option could be implemented as a separate utility; however, keeping the –k option would allow the option to be used in conjunction with other options in order to provide more flexibility for the user.

The –k option is not allowed to operate on access ACLs since access ACLs must always contain the required ACL entries corresponding to the file class permissions. if the –k option were to be allowed on an access ACL, then the target file could potentially be left an inconsistent state, i.e. with NO file permission bits.

The –k option does not report an error if the `setfacl` utility is used to remove a default ACL from a directory which does not contain a default ACL. This is done in order to avoid potentially generating many errors when the utility is used in conjunction with the `find` utility to remove all default ACL recursively from all directories within a filesystem hierarchy.

–n The –n option is used to indicate that a **mask** entry should not be generated nor should the permissions associated with the **mask** entry be recalculated by the `setfacl` utility. A user (or an application) can set a value for the mask which may restrict the permissions granted by additional ACL entries. The –n option allows the user to subsequently modify the ACL without automatically changing the **mask** entry and, thereby, inadvertently increasing the effective permissions of ACL entries. This option appears to be most useful when manipulating an ACL on a file whose permission bits are also being manipulated by `chmod`.

–m The –m option is used to update existing entries and to add new entries to the ACL.

–M The –M option is also used to update existing entries and to add new entries to an ACL. However, the –M option allows entries to be contained within a file (or to be obtained from standard input). This option is especially useful when using the `getfacl` utility in conjunction with the `setfacl` utility to copy an ACL from one object to another; the output from one utility can be piped directly into the input of the other. While this capability could be obtained using the shell's back quote substitution, this operation is expected to be frequent enough to justify an easy, direct method of specification.

–x The –x option is used to remove existing entries from an ACL. The letter “x” was chosen instead of “r” in order to not conflict with the use of “r” to indicate recursive operations in existing utilities.

–X The –X option is also used to remove existing entries from an ACL. However, like the –M option, the –X option allows entries to be contained within a file (or to be obtained from standard input).

Originally, the `setfacl` utility also contained the –i and –I options to completely replace an ACL with an entirely new ACL. These options were removed because this operation is effectively provided by the combination of the –b option with the –m option. Adding the additional options of –i and –I was viewed as adding unnecessary options and complexity to the utility.

The working group also considered specifying a single remove option with special operands:

–r entries This option removes the named ACL entries.

–r all This option removed all the ACL entries.

–r nonbase This option removed all the ACL entries except the base ACL entries.

The various –r options were considered to be inconsistent with other options to the utility and the syntax was considered to be overly complex. Separate options were chosen as being slightly less complex and better overall syntax.

It was suggested that the `getfacl` utility support the option of retrieving both the default and access ACL in one invocation of the utility. Specifically, it was suggested that this operation be the default operation of the utility and that the utility support the –a option for retrieving only the access ACL. While adding this feature to the utility is certainly a convenience for the user (instead of having to invoke `getfacl` twice), it also adds complexity to the use of the utility. This would be particularly apparent since the default ACL would only be retrieved when the utility was used to retrieve the ACLs on directories. Likewise, the option should actually be to retrieve all ACLs associated with the file (instead of just the default and access ACLs). Thus, any implementation defined ACLs associated with the file should also be retrieved. As a default operation, this would add significant complexity to the interface. Also, adding this option would make the `getfacl` utility less consistent with the `setfacl` utility unless `setfacl` were to be modified to accept both the default and access ACLs as input. This option would be best provided as an implementation defined extension to the `getfacl` utility.

It was suggested that the `setfacl` utility allow an option to recursively set an ACL throughout a filesystem hierarchy and to include an option to select the type of files to which the ACL would be applied. These options are not provided in the utility since the recursive selection of files based on type (as well as other criteria) is provided via the `find` utility. Thus, the recursive setting of an ACL on selected files can easily be accomplished via the combination of the `find` and `setfacl` utilities. This facility is not provided within `setfacl` in order to avoid the duplication of function in different utilities.

E.8.1.5 Evaluation Order of Option Characters

There are two general possibilities for the processing order of the options that may be specified in an invocation of the `setfacl` utility. The options may either be processed in a well defined order as specified by the standard or the options may be processed in the order of the occurrence of the options in the invocation of the utility. The standard could easily specify the apparent "most logical" evaluation order for the options (e.g., the –k and –b options first followed by the –m (–M) and –x (–X) options). However, the options are processed in the order in which they are specified by the user. This allows users the flexibility to determine the order of the options to best meet their needs.

E.8.1.6 ACL Entry Type Names

The working group considered using a lower case version of the programmatic ACL entry tag types as the ACL entry tag types for the first field of the external ACL entry representation, e.g., ACL_USER_OBJ => **user_obj**. This option has the advantage of being easier to parse, than using **user** for both the ACL_USER_OBJ and ACL_USER ACL entries. However, this option does not lead to easy aliases, e.g., **user_obj** is difficult to alias, but **user** can be aliased as *u*. The working group felt that the availability of easy aliases outweighs the complexity introduced in the parsing and decided not to use the lower case versions of the programmatic ACL entry tag types.

E.8.1.7 ACL Entry Permissions

The `setfacl` utility allows ACL entry permissions to be specified as a symbolic string with an absolute or relative value. The format of this symbolic string is different from the format used when specifying permissions in the *chmod*() utility. The format for the permissions was chosen to allow the output of the `getfacl` utility to be used without modification as input to the `setfacl` utility. This will allow a user to easily copy an ACL from one object to other objects. This additional functionality was believed to be more important than maintaining complete consistency with the format of the permissions in *chmod*().

E.8.1.8 Mask Entry Calculation

The `getfacl` and `setfacl` utilities are designed to make the existence of the ACL **mask** entry as transparent as possible to the users of the utilities. For most ACL manipulations, the permissions can be specified by the ACL entries and the permissions associated with the **mask** entry will be automatically calculated and reset to be the logical union of all of the permissions of all ACL entries in the file group class. While recalculation of the mask value is the default operation, the –n option is provided for the `setfacl` utility in order to allow users the ability to directly manipulate the mask value, as desired.

E.8.1.8.1 Mask Calculation Algorithm and Unsafe Conditions

In an earlier draft, the `setfacl` utility attempted to determine the caller's intentions in changing ACL entries and would warn the user of situations which the utility considered to be "unsafe" (i.e., when permissions could be granted to ACL entries which the utility determined that the user might not be expecting). In addition, the –p option was provided to actually remove such permissions from the ACL and the –c option to always recalculate the mask regardless of the utility's interpretation of the user's intentions. The mask recalculation algorithm attempted to detect conditions which the utility considered unsafe entries while minimizing, but not eliminating, false alarms. The algorithm also attempted to limit the detection of unsafe cases to situations which were considered to be highly unlikely user behavior.

The actual algorithm for detecting such “unsafe” conditions is described below for reference. Implementations may choose to add options which incorporate this (or a similar) algorithm in the `setfacl` utility. The algorithm specifies changes to the ACL, but those changes would only be made permanent in the ACL after all checks had been made and the operation determined to be “safe”. All operations are performed only on ACL entries that are members of the file group class.

The algorithm is specified below in programmatic form:

(1) Retrieve the existing ACL of the object.

(2) Perform all requests to remove entries and requests to reduce the permissions of existing entries.

(3) Calculate the union of the actual permissions of all remaining entries.

(4) Calculate the union of the effective permissions of all remaining entries.

(5) Determine which permissions differ between the actual and effective rights (logical XOR of results of steps 3 and 4).

(6) Perform all requests to add new entries to the ACL and all requests to increase the permissions of existing entries.

(7) Calculate the union of these newly granted permissions and the old effective permissions (step 4). This is the candidate new mask value.

(8) If there are any permissions in the candidate new mask that are also in the permissions that differ between the original actual and effective rights (step 5), applying the candidate new mask would unexpectedly grant some new right that the user did not intend. Unless the user specified one of the options –c, –p, or –n, this condition shall generate an error and the ACL will not be modified. If this condition does not hold, then apply the candidate new mask as the new mask.

The algorithm avoids false alarms that would occur if the new mask were simply calculated to be the logical OR of all the entries of the new file group class.

The following is an example of how false alarms could be avoided.

Consider an ACL with the following entries:

user::rwx

mask::r-x

user:user1:rwx #effective:r-x

user:user2:rwx #effective:r-x

group::r-x

other::---

If user1’s permissions were changed to r-x permission, a simple recalculation using *acl_calc_mask*() would result in changing the mask to rwx which would inadvertently grant w permission to user2. However, the algorithm specified above detects that removing user1’s w permission does not require altering the mask.

The –p option was provided to allow the user to remove ineffective permissions. The –c option was included in order to allow users to always request unconditional recomputation of the mask regardless of any unsafe conditions. As this option could be quite dangerous, it was suggested that an implementation issue a warning message if any unsafe conditions were detected.

Notice that the mask entry was only relevant when it had been “lowered” to actually reduce the permissions granted by one or more entries within the ACL. A “lowered” mask could only occur for three reasons. The first reason is that the mask may have been specified to a value less than some of the permissions in the default ACL (such as execute permission on a data file) when an object was created.

The second reason is that a program may have temporarily lowered the mask to lock out other users from the file.

The third reason is that the user may have lowered the mask using the *chmod*() utility explicitly.

In all cases, the user of the `setfacl` utility would need to know that the mask had been lowered, understand why the mask had been lowered, and would be required to be able to select the appropriate options for the utility in order to achieve the desired results.

Since it had been suggested that implementations issue warning messages for the detection of "unsafe" conditions, it was also suggested that such implementations also would provide a –s option that would suppress the messages.

The detection of “unsafe” conditions and the attempted interpretation of the user’s intention in manipulating the mask added significant complexity to the `setfacl` interface. This type of operation and the resulting interface were considered entirely too complex for users to understand or use effectively. As such, the –c and –p options were removed; and the default operation was changed to simply recalculate the mask. Likewise, the mask recalculation was changed to be simply the union of the permissions in all ACL entries within the file group class.

E.8.1.8.2 Mask Calculation and chmod

It was considered to allow the ACL mask entry to be set only by the `chmod` utility and not be modifiable by the `setfacl` utility. This restriction was rejected because it would have made copying ACLs from one file to another too difficult by requiring the use of the `chmod` utility as well as the ACL utilities.

E.10 Capability

E.10.1 Capability-Related Utilities

These utilities were determined to be the minimal set necessary for the determination and establishment of the capability attributes of files.

These is an argument that such utilities are administrative in nature and therefore outside of the scope of this working group. The working group noted, however, that installation scripts and programs are themselves portable applications that will need to work across implementations, and these utilities will be required to support them. In addition, should the argument be accepted, there are sufficient reasons for adopting standards for these types of utilities to persuade other existing working groups to just adopt them as part of their standard. Since the only practical result of not specifying these utilities is to merely delay their specification, we felt that they would best be specified by this committee.

Some responders to the initial ballot felt that there should be commands to assign capabilities to users as well. Since POSIX does not yet specify a user database or identification and authentication system, we felt that inclusion of such commands was premature. In addition, a few people proposed examples of systems which would conform to this standard where no capability data was directly associated with users. For these cases, requiring such commands would be an undue burden on the implementation. If an implementation does wish to assign capabilities to users, however, we believe that extending the syntax of the two commands presented here would be simple and straightforward.

E.10.1.1 Get and Set the Capability State of a Subject or Object

The `getfcap`, `getpcap` and `setfcap` utilities were included as a part of the standard primarily to support the definition of a standard, cross-implementation user interface for the administration of file capabilities. Increasingly, secure applications will need to have a standard means of being installed so that operators that do not necessarily have a strong background in security can just run a script supplied to them by a security administrator. In addition, one of the necessary functions of a security administrator is to periodically check the security related attributes of files and programs to ensure that they have not been tampered with. The standardization of utilities that support setting and display of file capability attributes is therefore considered to be necessary.

The grammar chosen for representing, setting and modifying capability states is a modified version of that used by `chmod` for the symbolic representation of mode bit operations. This representation was chosen because it is compact and is familiar to current users of POSIX systems.

It was decided not to add to the functionality of existing utilities or system functions in this area; specifically, the *stat*() system function and the `ls` utilities are already overburdened and complex.

E.11 Mandatory Access Control

E.11.1 General Overview

The following utilities have been added to support mandatory access control (MAC): `getfmac`, `setfmac`, and `getpmac`. These utilities were determined to be the minimal set necessary for the determination and establishment of MAC labels for files and processes. No utility is provided to allow users to modify the MAC label of an executing process for two reasons: (1) there is no precedent in POSIX.2 for utilities that modify the attributes (e.g., user ID or umask) of existing processes, and (2) no compelling argument has ever been put forward for why such a utility would be useful.

For each utility, the code is not derived from any existing system, and no source code was examined. None of these interfaces depend on a precise definition of what constitutes a MAC label provided. That is, the format of the MAC label argument to `setfmac` is not specified, nor is the format of the MAC labels written to standard out by `getfmac` and `getpmac`. This is because POSIX.1e does not constrain implementations in terms of the allowable human-readable representations of MAC labels. (As a practical matter, most *implementations* probably should not constrain them: the human-readable representations of MAC labels will typically be administrator defined.)

Because the precise format of the text representation of MAC labels is not specified, both `getfmac` and `getpmac` only loosely specify the location of the label within the output stream. Until standards for label syntax are specified, utilities (especially standard utilities such as `awk` or `grep`) cannot parse the output of these utilities. Therefore, the output of these utilities are primarily useful only for display to users. Specifying a standard for the syntax of the text representation of labels was considered, but rejected for inclusion in this standard.

Note that conforming implementations may choose to provide a more rigorously specified output format to assist implementation-specific parsing utilities, or provide a visually more easily understood output format through the use of an additional argument.

E.11.2 Separate Utilities

The working group considered adding the functionality specified in the `getfmac` to `ls`. However, the working group strongly feels that the `ls` interface is already sufficiently complex and that adding MAC label display capabilities to `ls` would further complicate an overly complex interface.

The working group also considered designing a single utility interface that included all of the utility interfaces specified in this standard. However, one of the goals of the working group is to produce a modular set of interfaces. Since the working group felt that this solution does not fit into a modular model, the group discarded this solution.

E.11.3 Label Input and Output

As noted above, the format of the labels produced by `getfmac`, `setfmac`, and `getpmac` are not precisely defined. Nevertheless, in order to provide support for portable applications, it was felt that these utilities should be required to interoperate. This is necessary to allow portable applications to use the output of the utilities, and portable shell scripts to reuse the labels output from the utilities (e.g., in save-alter-restore algorithms that temporarily modify file labels). Therefore, the specification requires that the labels output by `getfmac` and `getpmac` must be in a format suitable for re-input to the `setfmac` utility. Similarly, the labels output by `getfmac` and `getpmac` must be suitable for re-input to the *mac_from_text()* function defined in section 26.3.7 of POSIX.1e. Finally, we require that the labels produced by the *mac_to_text()* defined in section 26.3.17 of POSIX.1e. must be suitable for re-input to the `setfmac` utility.

E.11.4 Utility use of Capabilities

Actual implementation of utilities such as `setfmac` may require the utility to possess appropriate privilege to perform its function, but this standard is mute on whether privilege is required or the specific capabilties which may be required. This is an interface specification for the utilities. As such, the interface to these utilities must be specified, as must their behavior. Their implementation, however, is outside the scope of this standard. A conforming implementation could certainly implement these utilities without using POSIX interfaces. Whatever (native) interfaces are used to implement the utility may not require capabilities, or if they do, they may not require POSIX capabilities. Therefore, specifying that capabilities are required may, in at least some cases, be incorrect.

E.12 Information Labeling

E.12.1 General Overview

The following utilities have been added to support information labeling: `getfinf`, `setfinf`, and `getpinf`. These utilities were determined to be the minimal set necessary for the determination and establishment of information labels for files and processes. No utility is provided to allow users to modify the information label of an executing process for two reasons: (1) there is no precedent in POSIX.2 for utilities that modify the attributes (e.g., user ID or umask) of existing processes, and (2) no compelling argument has ever been put forward for why such a utility would be useful.

For each utility, the code is not derived from any existing system, and no source code was examined. None of these interfaces depend on a precise definition of what constitutes an information label. That is, the format of the information label argument to `setfinf` is not specified, nor is the format of the information labels written to standard out by `getfinf` and `getpinf`. This is because POSIX.1e does not constrain implementations in terms of the allowable human-readable representations of information labels. (As a practical matter, most *implementations* probably should not constrain them: the human-readable representations of information labels will typically be administrator defined.)

Because the precise format of the text representation of information labels is not specified, both `getfinf` and `getpinf` only loosely specify the location of the label within the output stream. Until standards for label syntax are specified, utilities (especially standard utilities such as `awk` or `grep`) cannot parse the output of these utilities. Therefore, the output of these utilities are primarily useful only for display to users. Specifying a standard for the syntax of the text representation of labels was considered, but rejected for inclusion in this standard.

Note that conforming implementations may choose to provide a more rigorously specified output format to assist implementation-specific parsing utilities, or provide a visually more easily understood output format through the use of an additional argument.

E.12.2 Separate Utilities

The working group considered adding the functionality specified in the `getfinf` to `ls`. However, the working group strongly feels that the `ls` interface is already sufficiently complex and that adding information label display capabilities to `ls` would further complicate an overly complex interface.

The working group also considered designing a single utility interface that included all of the utility interfaces specified in this standard. However, one of the goals of the working group is to produce a modular set of interfaces. Since the working group felt that this solution does not fit into a modular model, the group discarded this solution.

E.12.3 Label Input and Output

As noted above, the format of the labels produced by `getfinf`, `setfinf`, and `getpinf` are not precisely defined. Nevertheless, in order to provide support for portable applications it was felt that these utilities must interoperate both with themselves, and with the relevant functions defined in section 27 of POSIX.1e. This is necessary to allow portable applications to use the labels produced by the utilities, portable shell scripts to use the labels output by applications using the applicable IL functions and to reuse the labels output from the utilities (e.g., in save-alter-restore algorithms that temporarily modify file labels), etc. Therefore, the specification requires that the labels output by `getfinf` and `getpinf` must be in a format suitable for re-input to the `setfinf` utility and to the *inf_from_text()* function defined in section 27.3.9 of POSIX.1e. Similarly, we require that the labels produced by the *inf_to_text()* function defined in section 27.3.17 of POSIX.1e must be suitable for re-input to the `setfinf` utility.

Annex F
(informative)
Ballot Instructions

This annex will not appear in the final standard. It is included in the draft to provide instructions for balloting that cannot be separated easily from the main document, as a cover letter might.

It is important that you read this annex, whether you are an official member of the PSSG Balloting Group or not; comments on this draft are welcomed from all interested technical experts.

Summary of Draft 17 Instructions

This is a recirculation on the P1003.2c ballot. The procedure for a recirculation is described in this annex. Because this is a recirculation comments may only be provided concerning sections that have changed, sections affected by those changes, or on rejected comments from the previous ballot.

Send your ballot and/or comments to:

IEEE Standards Office
Computer Society Secretariat
ATTN: PSSG Ballot (Carol Buonfiglio)
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331

It would also be very helpful if you sent us your ballot in machine-readable form. Your official ballot must be returned via mail to the IEEE office; if we receive only the e-mail or diskette version, that version will not count as an official document. However, the online version would be a great help to ballot resolution. Please send your e-mail copies to the following address:

casey@sgi.com

or you may send your files in ASCII format on DOS 3.5 inch formatted diskettes (720Kb or 1.4Mb), or Sun-style QIC-24 cartridge tapes to:

Casey Schaufler
Silicon Graphics
2011 North Shoreline Blvd.
P.O. Box 7311
Mountain View, CA 94039-7311

Background on Balloting Procedures

The Balloting Group consists of approximately eighty technical experts who are members of the IEEE or the IEEE Computer Society; enrollment of individuals in this group has already been closed. There are also a few "parties of interest" who are not members of the IEEE or the Computer Society. Members of the Balloting Group are required to return ballots within the balloting period. Other individuals who may happen to read this draft are also encouraged to submit comments concerning this draft. The only real difference between members of the Balloting Group and other individuals submitting ballots is that *affirmative* ballots are only counted from Balloting Group members who are also IEEE or Computer Society members. (There are minimum requirements for the percentages of ballots returned and for affirmative ballots out of that group.) However, objections and nonbinding comments must be resolved if received from any individual, as follows:

(1) Some objections or comments will result in changes to the standard. This will occur either by the republication of the entire draft or by the publication of a list of changes. The objections/comments are reviewed by a team from the POSIX Security working group, consisting of the Chair, Vice Chair, Technical Editor, and a group of Technical Reviewers. The Chair will act as the Ballot Coordinator. The Technical Reviewers each have subject matter expertise in a particular area and are responsible for objection resolution in one or more sections.

(2) Other objections/comments will not result in changes.

   (a) Some are misunderstandings or cover portions of the document (front matter, informative annexes, rationale, editorial matters, etc.) that are not subject to balloting.

   (b) Others are so vaguely worded that it is impossible to determine what changes would satisfy the objector. These are referred to as *Unresponsive*. (The Technical Reviewers will make a reasonable effort to contact the objector to resolve this and get a newly worded objection.) Further examples of unresponsive submittals are those not marked as either *Objection*, *Comment*, or *Editorial*; those that do not identify the portion of the document that is being objected to (each objection must be separately labeled); those that object to material in a recirculation that has not changed and do not cite an unresolved objection; those that do not provide specific or general guidance on what changes would be required to resolve the objection.

(c) Finally, others are valid technical points, but they would result in decreasing the consensus of the Balloting Group. (This judgment is made based on other ballots and on the experiences of the working group through over seven years of work and fifteen drafts preceding this one.) These are referred to as *Unresolved Objections*. Summaries of unresolved objections and their reasons for rejection are maintained throughout the balloting process and are presented to the IEEE Standards Board when the final draft is offered for approval. Summaries of all unresolved objections and their reason for rejection will also be sent to members of the Balloting Group for their consideration upon a recirculation ballot. (Unresolved objections are not circulated to the ballot group for a re-ballot.) Unresolved objections are only circulated to the balloting group when they are presented by members of the balloting group or by parties of interest. Unsolicited correspondence from outside these two groups may result in draft changes, but are not recirculated to the balloting group members.

Please ensure that you correctly characterize your ballot by providing one of the following:

(1) Your IEEE member number

(2) Your IEEE Computer Society affiliate number

(3) If (1) or (2) don't apply, a statement that you are a "Party of Interest"

Ballot Resolution

The general procedure for resolving ballots is:

(1) The ballots are put online and distributed to the Technical Reviewers.

(2) If a ballot contains an objection, the balloter may be contacted individually by telephone, letter, or e-mail and the corrective action to be taken described (or negotiated). The personal contact will most likely not occur if the objection is very simple and obvious to fix or the balloter cannot be reached after a few reasonable attempts. Repeated failed attempts to elicit a response from a balloter may result in an objection being considered unresponsive, based on the judgment of the Ballot Coordinator. Once all objections in a ballot have been resolved, it becomes an affirmative ballot.

(3) If any objection cannot be resolved, the entire ballot remains negative.

(4) After the ballot resolution period the technical reviewers may chose to either *re-ballot* or *recirculate* the ballot, based on the status of the standard and the number and nature of outstanding (i.e., rejected or unresolved) objections. The ballot group may or may not be reformed at this time. If a *reballot* is chosen, the entire process of balloting begins anew. If a *recirculation* is chosen, only those portions affected by the previous ballot will be under consideration. This ballot falls into this latter category

(5) On a *recirculation* ballot, the list of unresolved objections, along with the ballot resolution group's reasons for rejecting them will be circulated to the existing ballot group along with a copy of the document that clearly indicates all changes that were made during the last ballot period. You have a minimum of ten days (after an appropriate time to ensure the mail got through) to review these two documents and take one of the following actions:

    (a) Do nothing; your ballots will continue to be counted as we have classified them, based on items (3) and (4).

    (b) Explicitly change your negative ballot to affirmative by agreeing to remove all of your unresolved objections.

    (c) Explicitly change your affirmative ballot to negative based on your disapproval of either of the two documents you reviewed. If an issue is not contained in an unresolved objection or is not the result of a change to the document during the last ballot resolution period, it is not allowed. Negative ballots that come in on recirculations cannot be cumulative. They shall repeat any objections that the balloter considers unresolved from the previous recirculation. Ballots that simply say "and all the unresolved objections from last time" will be declared unresponsive. Ballots that are silent will be presumed to fully replace the previous ballot, and all objections not mentioned on the most current ballot will be considered as successfully resolved.

(6) Rather than reissue the entire document, a small number of changes may result in the issuance of a change list rather than the entire document during recirculation.

(7) A copy of all your objections and our resolutions will be mailed to you.

(8) If at the end of a recirculation period there remain greater than seventy-five percent affirmative ballots, and no new objections have been received, a new draft is prepared that incorporates all the changes. This draft and the unresolved objections list go to the IEEE Standards Board for approval. If the changes cause too many ballots to slip back into negative status, another resolution and recirculation cycle begins.

Balloting Guidelines

This section consists of guidelines on how to write and submit the most effective ballot possible. The activity of resolving balloting comments is difficult and time consuming. Poorly constructed comments can make that even worse.

We have found several things that can be done to a ballot that make our job more difficult than it needs to be, and likely will result in a less than optimal response to ballots that do not follow the form below. Thus it is to your advantage, as well as ours, for you to follow these recommendations and requirements.

If a ballot that significantly violates the guidelines described in this section comes to us, we may determine that the ballot is unresponsive.

If we recognize a ballot as "unresponsive," we will try to inform the balloter as soon as possible so he/she can correct it, but it is ultimately the balloter's responsibility to assure the ballot is responsive. Ballots deemed to be "unresponsive" may be ignored in their entirety.

Some general guidelines to follow before you object to something:

(1) Read the Rationale section that applies to the troublesome area. In general there is a matching informative section in the Rationale Annex for each normative section of the standard. This rationale often explains why choices were made and why other alternatives were not chosen.

(2) Read the Scope, section 1, to see what subset of functionality we are trying to achieve. This standard does not attempt to be everything you ever wanted for accomplishing secure software systems. If you feel that an additional area of system interface requires standardization, you are invited to participate in the security working group which is actively involved in determining future work.

(3) Be cognizant of definitions in section 2. We often rely in the document on a precise definition from section 2 which may be slightly different than your expectation.

Typesetting is not particularly useful to us. Also please do not send handwritten ballots. Typewritten (or equivalent) is fine, and if some font information is lost it will be restored by the Technical Editor in any case. You may use any word processor to generate your objections but do not send `[nt]roff` (or any other word processor) input text. Also avoid backslashes, leading periods and apostrophes in your text as they will confuse our word processor during collation and printing of your comments. The ideal ballot is formatted as a "flat ASCII file," without any attempt at reproducing the typography of the draft and without embedded control characters or overstrikes; it is then printed in Courier (or some other typewriter-like) font for paper-mailing to the IEEE Standards Office and simultaneously e-mailed to the Working Group Ballot Coordinator at the following email address.

casey@sgi.com

Don't quote others' ballots. Cite them if you want to refer to another's ballot. If more than one person wants to endorse the same ballot, send just the cover sheets and one copy of the comments and objections. [Note to Institutional Representatives of groups like X/Open, OSF, UI, etc.: this applies to you, too. Please don't duplicate objection text with your members.] Multiple identical copies are easy to deal with, but just increase the paper volume. Multiple almost-identical ballots are a disaster, because we can't tell if they are identical or not, and are likely to miss the subtle differences. Responses of the forms:

— "I agree with the item in <someone>'s ballot, but I'd like to see this done instead"

— "I am familiar with the changes to `foo` in <someone>'s ballot and I would object if this change is [or is not] included" are very useful information to us. If we resolve the objection with the original balloter (the one whose ballot you are referencing), we will also consider yours to be closed, unless you specifically include some text in your objection indicating that should not be done.

Be very careful of “Oh, by the way, this applies <here> too” items, particularly if they are in different sections of the document that are likely to be seen by different reviewers. They are probably going to be missed! Note the problem in the appropriate section, and cite the detailed description if it’s too much trouble to copy it. The reviewers don’t read the whole ballot. They only read the parts that appear in the sections that they have responsibility for reviewing. Particularly where definitions are involved, if the change really belongs in one section but the relevant content is in another, please include two separate comments/objections.

Please consider this a new ballot that should stand on its own. Please do not make backward references to your ballots for the previous draft. Include all the text you want considered here, because the Technical Reviewer will not have your old ballot. (The old section and line numbers won’t match up anyway.) If one of your objections was not accepted exactly as you wanted, it may not be useful to send in the exact text you sent before; read our response to your objection (you will receive these in a separate mailing) and the associated Rationale section and come up with a more compelling (or clearly-stated) justification for the change.

Please be very wary about global statements, such as “all of the arithmetic functions need to be defined more clearly.” Unless you are prepared to cite specific instances of where you want changes made, with reasonably precise replacement language, your ballot will be considered unresponsive.

Ballot Form

The following form is strongly recommended. We would greatly appreciate it if you sent the ballot in electronic form in addition to the required paper copy. Our policy is to handle all ballots online, so if you don’t send it to us that way, we have to type it in manually. See the first page of this Annex for the addresses and media. As you’ll see from the following, formatting a ballot that’s sent to us online is much simpler than a paper-only ballot.

The paper ballot should be page-numbered, and each page should contain the name, e-mail address, and phone number(s) of the objector(s). The electronic copy of the ballot should only have it once, in the beginning. Please leave adequate (at least one inch) margins on both sides.

Don’t format the ballot as a letter or document with its *own* section numbers. These are simply confusing. As shown below, it is best if you cause each objection and comment to have a sequential number that we can refer to amongst ourselves and to you over the phone. Number sequentially from 1 and count objections, comments, and editorial comments the same; don’t number each in its own range.

We recognize three types of responses:

Objection A problem that must be resolved to your satisfaction prior to your casting an "affirmative" vote for the document.

Comment A problem that you might want to be resolved by the reviewer, but which does not in any way affect whether your ballot is negative or positive. Any response concerning the pages preceding page 1 (the Front matter), Rationale text with shaded margins, Annexes, NOTES in the text, footnotes, or examples will be treated as a non-binding comment whether you label it that way or not. (It would help us if you'd label it correctly.)

Editorial A problem that is strictly an editorial oversight and is not of a technical nature. Examples are: typos; misspellings; English syntax or usage errors; appearances of lists or tables; arrangement of sections, clauses, and subclauses (except where the location of information changes the optionality of a feature).

To help us in our processing of your objections and comments, we are requiring that all comments, objections and editorial comments meet the following specific format. (We know that the format defined below contains redundant information but it has become a de facto standard used by many different POSIX standard ballots. It is felt that it is better to continue to use this format with the redundancies rather than to create a new format just for 1003.1e and P1003.2c)

Separate each objection/comment with a line of dashes ("-"), e.g.,

```
--------------------------------------------------------------------------------
```

Precede each objection/comment with two lines of identifying information:

The first line should contain:

@ *<section>*.*<clause>* *<code>* *<seqno>*

where:

@ At-sign in column 1 (which means no @'s in any other column 1's).

<section> The major section (chapter or annex) number or letter in column 3. Use zero for Global or for something, like the frontmatter, that has no section or annex number.

<clause> The clause number (second-level header). Please do not go deeper than these two levels. In the text of your objection or comment, go as deep as you can in describing the location, but this code line uses two levels only.

<code> One of the following lowercase letters, preceded and followed by spaces:

o Objection.

c Comment.

e Editorial Comment.

<seqno> A sequence number, counting all objections and comments in a single range.

The second line should contain:

```
<seqno>. Sect <sectno> <type>. page <pageno>, line <lineno>:
```

where:

<seqno> The sequence number from the preceding line

<sectno> The full section number. (Go as deep as you can in describing the location.)

<type> One of the following key words/phrases, preceded and followed by spaces:

```
OBJECTION
COMMENT
EDITORIAL COMMENT
```

<pageno> The page number from the document.

<lineno> The line number or range of line numbers that the object/comment relates to.

For each objection, comment, or editorial comment, you should provide a clear statement of the problem followed by the action required to solve that problem.

`Problem:`

A clear statement of the problem that is observed, sufficient for others to understand the nature of the problem. (Note that you should identify problems by section, page, and line numbers. This may seem redundant, but if you transpose a digit pair, we may get totally lost without a cross-check like this. Use the line number where the problem starts, not just where the section itself starts; we sometimes attempt to sort objections by line numbers to make editing more accurate. If you are referring to a range of lines, please don't say "lines 10xx;" use a real range so we can tell where to stop looking. Please try to include enough context information in the problem statement (such as the name of the function or command) so we can understand it without having the draft in our laps at the time. (It also helps you when we e-mail it back to you.)

`Action:`

A precise statement of the actions to be taken on the document to resolve the objection above, which if taken verbatim will completely remove the objection.

If there is an acceptable range of actions, any of which will resolve the problem for you if taken exactly, please indicate all of them. If we accept any of these, your objection will be considered as resolved.

If the Action section is omitted or is vague in its solution, the objection may be reclassified as a nonbinding comment. The Technical Reviewers, being human, will give more attention to Actions that are well-described than ones that are vague or imprecise. The best ballots of all have very explicit directions to substitute, delete, or add text in a style consistent with the rest of the document, such as:

```
Delete the sentence on lines 101-102:

"The implementation shall not ... or standard error."

On line 245, change "shall not" to "should not".

After line 711, add:

-c  Calculate the mask permissions and update the mask.
```

Some examples of poorly-constructed actions:

```
Remove all features of this command that are not supported by BSD.

Add -i.

Make this command more efficient and reliable.

Use some other flag that isn’t so confusing.

I don’t understand this section.

Specify a value--I don’t care what.
```

Sample Response:

```
Joseph Balloter                 (999)123-4567       page  4 of 17.
EMAIL: jmb@mycomp.com     FAX:  (999)890-1234

----------------------------------------------------------------

@ 1.1 o 23
23. Sect 1.1 OBJECTION. page 7, line 9:

Problem:

The current draft describes one the mechanisms specified in it as
"Least Privilege" which is incorrect.  "Least Privilege" is a
general principle related to access control rather than a mechanism.
In fact, the definition given in the standard (p.  91, l.  274)
calls it a principle rather than a mechanism.

Action:

Replace line 9 with:  "(3)  Enforcement of Least Privilege"

----------------------------------------------------------------

@ 3.1 o 24
24. Sect 3.1 OBJECTION. page 27, line 13:

Problem:

"during process of changing ACL" is vague.
Could be read as the duration from acl_read through acl_write.

Action:
```

Should state "while ACL is being written (acl_write)".

@ 3.3 e 25
25. Sect 3.3.1 EDITORIAL COMMENT. page 29, line 68:

Problem:

The two previous sentences describe the "ACL_USER_OBJ entry" and the "ACL_GROUP_OBJ entry". Line 68 describes "ACL_OTHER_OBJ", the word "entry" should be added for consistency.

Action:

change "ACL_OTHER_OBJ" to "ACL_OTHER_OBJ entry"

Sample Response (continued):

```
Joseph Balloter                  (999)123-4567       page  5 of 17.
EMAIL: jmb@mycomp.com     FAX:   (999)890-1234

----------------------------------------------------------------

@ 4.5 c 26
26. Sect 4.5.1.1 COMMENT. page 92, line 836:

Problem:

There is no introduction to table 4-1.

Action:

Add before line 836  "The aud_ev_info_t structure shall contain at
least the following fields:"

----------------------------------------------------------------

@ 6.5 o 27
27. Sect 6.5.7.2 OBJECTION. page 181, line 449-450:

Problem:

Can this "must" be tested ?
Is this really needed since the format of the label is undefined
and no functions are provided to access the individual components
(so that a comparison could be made). This seems to be a comment
that could just as easily be applied to most other mac functions,
say mac_freelabel for example.

Action:

Suggest either moving this into the MAC introductory section,
striking or changing "must" to "should" or "are advised".

----------------------------------------------------------------
```

Thank you for your cooperation and assistance in this important balloting process.

Lynne M. Ambuel
Chair, POSIX Security Working Group

Identifier Index

Topical Index

Contents