

IEEE Std 1003.1-1988/INT, 1992 Edition

IEEE Standards Interpretations for IEEE Standard Portable Operating System Interface for Computer Environments (IEEE Std 1003.1-1988)

Circuits and Devices

Communications Technology

IEEE Computer Society

Sponsored by the
Technical Committee on Operating Systems and Application Environments

*Electromagnetics and
Radiation*

Energy and Power

Industrial Applications

*Signals and
Applications*

*Standards
Coordinating
Committees*

IEEE Std 1003.1-1988/INT, 1992 Edition



Published by the Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA.

July 8, 1992

SH15313

AVA Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 06, 2026 at 04:07:19 UTC from IEEE Xplore. Restrictions apply.

IEEE Standards Interpretations

for

**IEEE Standard Portable Operating System
Interface for Computer Environments
(IEEE Std 1003.1-1988)**

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

Abstract: The Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society carried out a series of analyses of various problems encountered by users of IEEE Std 1003.1-1988, IEEE Standard Portable Operation System Interface for Computer Environments. The results of its deliberations are presented in this document. The intent is to give the POSIX community reasonable ways of interpreting the unclear portions of the standard.

Keywords: computer, computer environments, interfaces, portable operating system interface, POSIX

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright ©1992 by the
Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1992
Printed in the United States of America

ISBN 1-55937-216-8

*No part of this publication may be reproduced in any form,
in an electronic retrieval system or other wise,
without the prior written permission of the publisher.*

IEEE Standards Interpretations

Occasionally, questions may arise regarding the meaning of portions of standards as they relate to specific applications. Such requests for interpretations should ask for clarifications of the exact nature of the contents of the standard.

Questions relating to such interpretations are reviewed and evaluated by a balance of members representing the specific committee interests. This officially formed interpretations subgroup creates and circulates a draft interpretation among its members. It transmits a final interpretation to the party initiating the request only after consensus has been achieved. The interpretation is also given to the standards-developing committee to consider addressing it in a supplement or the next revision to the standard.

Interpretations are issued to explain and clarify the intent of the standard and are not intended to constitute an alteration to the original standard or to supply consulting information. The interpretations subgroup cannot make new rules to fit situations not yet covered in the standard, even if the investigations of the subgroup lead it to conclude that the requirement is incomplete or in error. Changes to the standard are made only through revisions or supplements to the standard.

It is recognized that requests are frequently received that are partially or totally requests for information rather than requests for an interpretation. It is inappropriate to issue an official interpretation to answer such requests. The interpretations subgroup may, however, find from its research that the literal printing of the standards text is not identical to that approved by the standards developers and may issue an editorial correction as a part of its interpretation.

The original interpretations requests in this document have been lightly edited to remove extraneous matter and to focus on the problem presented. Some illustrations have been redrawn for publication. With these exceptions, requests are in the form received.

Procedure for Requesting an Interpretation

Requests for interpretations should be addressed to

Secretary, IEEE Standards Board
IEEE Standards Department
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331

Requests for interpretations should include

- (1) The specific designation of the standard, including the year of publication.
- (2) The specific subsection being questioned.
- (3) The applicable conditions for the case in question.

Line drawings should be black ink or excellent black pencil originals. Photos should be black-and-white glossy prints. These illustrations must be reproduced for committee circulation and eventually will be used to supplement the text of the next edition. Clear diagrams and pictures will make the work of interpretation easier and more valuable to users.

Requests, including all supplementary material, must be in a form that is easily reproducible. If suitable for consideration, requests will be sent to the interpretations subgroup. After consideration by the subcommittee, which may involve many exchanges of correspondence, the inquirer will be notified of the decision of the subgroup. Decisions will be published from time to time in cumulative form and may be ordered from the IEEE.

Foreword

[This Foreword is not a part of IEEE Std 1003.1-1988/TNT (1992 Edition), IEEE Standards Interpretations for IEEE Standard Portable Operating System Interface for Computer Environments.]

After approval and publication of a standard, questions may arise regarding the meaning of portions of standards as they relate to users of the standard. In the case of IEEE Std 1003.1-1988, these questions arose from both system implementors and application authors. When a need for interpretation is brought to the attention of the IEEE, the Institute initiates action to prepare appropriate responses.

IEEE Std 1003.1-1988 was developed by the Technical Committee on Operating Systems (TCOS) of the IEEE Computer Society. Each interpretation request is forwarded to the members of the TCOS interpretations group and a subgroup is invited to generate a draft interpretation for review. The draft interpretation is made available to the whole interpretations group for review and comment. When consensus has been reached within the interpretations group, the IEEE transmits the final interpretation to the party initiating the request.

It is recognized that requests are frequently received that are partially or totally requests for information rather than requests for interpretation. The IEEE considers it inappropriate to issue an official interpretation to answer such requests. This includes requests for advice as to the means by which particular facilities described in the standard could be implemented.

Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. The revision process takes input from a wide variety of sources in order to develop a revised standard. One of these sources is the interpretations that have been issued against the extant version of the standard. It is not to be expected that the text of the interpretation will appear in the revised standard, but rather that the revision will address the issues raised by the interpretation request through the consensus process. The revision process is allowed to resolve ambiguities in the original standard through textual change; this capability is not allowed within the interpretations process.

In the case of this publication, IEEE Std 1003.1-1988 was revised in 1990 to become IEEE Std 1003.1-1990 (ISO/IEC 9945-1: 1990). As such, the interpretations committee reviewed these interpretations prior to publication to determine if they applied to the 1990 edition of the standard as well as the 1988 edition. If not, information was provided on whether the text of the 1990 edition was changed from that in the 1988 (since revisions to the 1990 edition could have incorporated the recommendations of the interpretations committee).

This document contains a compilation of all of the interpretations completed against the IEEE Std 1003.1-1988. The IEEE will provide details of interpretation requests received that have not yet appeared in a publication. Where a formal interpretation has been completed, this will include the text of the interpretation.

At the time that this compilation of interpretations was published, the following members had participated in the IEEE Std 1003.1-1988 interpretations group:

Standards Subcommittee for TCOS

Chair:	Jim Isaak
Treasurer:	Quin Hahn
Secretary:	Shane McCarron
Vice-Chair, Interpretations:	Robert Bismuth (to April 1992) Andrew Twigger

Robert Bismuth
Dave Decot
Jeff Kimmel
Robert Lenk
Shane McCarron

Jim McGuinness
Roy McKean
Martha Nalebuff
Paul Rabin

John Reed
Seth Rosenthal
Keith Stuck
Andrew Twigger
David Willcox

Mary Lynne Nielsen
IEEE Standards Department Project Editor



Documents
Document Sharing Hub

Interpretation Number: 1
Topic: *setlocale*(LC_ALL) return strings
Relevant Sections: 8.1.2, 8.1.2.2
Applies to 1990 edition: No
Related text in 1990 revised: Yes

Interpretation Request

The following discussion is based on the definition of the *setlocale*() function as specified in the May 13, 1988 Draft C Standard and as extended by IEEE Std 1003.1-1988.

The description of the return value of the *setlocale*() function is not fully defined when the function is invoked in the following manner:

```
setlocale(LC_ALL, "");
```

Given that all the relevant environment variables are valid, *setlocale*() will set the environment to reflect the values of the environment variables. One could conceive that portions of the environment may have differing values corresponding to LC_NUMERIC, LC_TIME, LC_COLLATE, LC_MONETARY, etc., to accommodate a mixed international environment. The return value for this scenario is left undefined by the standard (i.e., it is not clear which of the LC_* values to return).

The return value of the *setlocale*() function is also not defined when the function is invoked as follows:

```
setlocale(LC_ALL, (char *)NULL);
```

Given that each category in the environment is set to a different string, it is not clear (or defined) which string shall be returned.

IEEE Interpretation for 1003.1-1988

The requestor has asked for an interpretation regarding *setlocale*() on two counts: What happens when *setlocale*() is called with LC_ALL as the category and a locale of "" and what happens when *setlocale*() is called with LC_ALL as the category and a locale of NULL.

First, it is important to note that most of the intelligence regarding *setlocale*() comes from the Draft C Standard (October 31, 1988 draft), and that IEEE Std 1003.1-1988 only provides some additional information regarding interaction of *setlocale*() and environment variables in a POSIX.1 implementation.

The Draft C Standard clearly specifies that *setlocale*(LC_ALL, "") shall set the implementation-defined native environment. In the case of POSIX.1 implementations, this means a locale that is specified by the user via environment variables or by the implementation in the absence of these environment variables. If execution of this function completes successfully, it shall return a pointer to an unspecified string representing the various categories of a heterogeneous locale.

The Draft C Standard also clearly specifies that "a null pointer for **locale** causes the **setlocale** function to return a pointer to the string associated with the **category** for the program's current locale; the program's locale is not changed." It is safe to assume that this is true for the LC_ALL "category" as well as the others. It will return an unspecified format string representing the current setting of this category, which corresponds to an explicit, valid user setting, or to the setting of the environment variables or an implementation-defined default locale in the absence of explicit settings. The contents of this string, while unspecified, will represent the various categories of a heterogeneous locale.

The interpretations committee believes that no further interpretation or P1003.1 committee action is necessary on this matter.

Rationale for Interpretation

No additional rationale has been provided for this interpretation.

Interpretation Number: 2
Topic: *cuserid()*
Relevant Sections: 4.2.4.2, 4.2.4.3
Applies to 1990 edition: No
Related text in 1990 revised: Yes

Interpretation Request

In 4.2.4.2, the *cuserid()* function is described as a function returning “a name associated with the effective user ID of the process.” Later in the same section, it is stated that “the *cuserid()* function generates a character representation of the login name of the owner of the current process.” In 4.2.4.3, describing the return value of the *cuserid()* function, it is stated that “if the login name cannot be found, *cuserid()* returns a **NULL** pointer.” The login name is effectively defined by the *getlogin()* function when 4.2.4.2 says: “The *getlogin()* function returns a pointer to the user’s login name.” Earlier in the same section, it clarifies what the login name is when it says: “. . . the *getlogin()* function returns the name associated by the login activity with the control terminal.” From the description of *getlogin()*, it seems that the login name is not necessarily the same as the name associated with the effective user ID. From the description of *cuserid()*, it is not clear whether the login name or the name associated with the effective user ID is returned by *cuserid()*.

If the name associated with the effective user ID is to be returned, it is not clear why the POSIX committee has chosen to depart from the historical use of the *cuserid()* function as implemented in System V (i.e., *cuserid()* is equivalent to calling *getlogin()* and, if *getlogin()* fails, calling *getpwuid*[*getuid()*]). Note that if “the recommended procedure is either to call the *cuserid()* function, or to call *getlogin()* and, if it fails, to call the *getpwuid()* function with the value returned by the *getuid()* function” (see 4.2.4.2), differing results may occur if the *cuserid()* function relies on the effective user ID in determining the name to return. The rationale does not explain why this deviation was chosen.

To further the confusion, it is stated later in 4.2.4.3 that “the *cuserid()* function may use the *getpwnam()* function . . .” This statement begs the question of whether the effective user ID or the real user ID shall be used in determining the return value for the *cuserid()* function.

IEEE Interpretation for 1003.1-1988

The *cuserid()* function is always required to return a name associated in the user database with the effective user ID of the calling process.

Rationale for Interpretation

The first paragraph of 4.2.4.2 clearly and unambiguously requires that *cuserid()* return a name associated with the effective user ID of the calling process. The fifth paragraph implies that the name is associated with that user ID by the user database.

The first paragraph almost completely defines this function’s behavior. The second paragraph provides only a recommended practice and does not actually describe behavior. The statement in the fifth paragraph using the terms “login name” and “owner of the current process” could be ambiguous standing alone because those terms are not clearly defined by the standard. The terms become unambiguous in the context of the first paragraph. The owner of the process is the user with a user ID matching the effective user ID as the process. The login name is a name associated with that user ID by the user database. While “login name” is used somewhat differently in this section to describe a name associated by the login activity, all such descriptions are specifically applied only to the *getlogin()* function and not to the *cuserid()*

function. Since there is no clear way to associate one login activity with a user ID, this cannot be the meaning in this context. Section 2.7, under the description of the LOGNAME environment variable, describes a login name as a name in the user database. This meaning clearly fits the rest of the definition of the function.

Notes from meetings of the working group show that the intent was to have *cuserid()* and *getlogin()* be distinct functions with distinct purposes. The *cuserid()* function was to provide a name for the current effective user ID for utilities that need it, while *getlogin()* was to provide the name under which the user logged in. The System V semantics, where *cuserid()* is a closer variation on *getlogin()*, were not seen as clearly useful.

The statement in 4.2.4.3 that “the *cuserid()* function may use the *getpwnam()* function” does not “beg the question,” or even relate to the question, of which user ID is used by *cuserid()*. The statement is totally orthogonal to that question, and thus does not alter the interpretation that the name returned is associated with the effective user ID. That statement is simply a warning to application writers that the two functions may share and thus overwrite the same static data space. The statement should probably mention *getpwuid()*, since it would be a more appropriate function to use than *getpwnam()*; however, the point about static data area is the same.

The current working draft of a revision to the standard makes several changes to the wording that change the use of the term “login name” and otherwise clarify the wording consistently with this interpretation.

Interpretation Number: 3
Topic: Namespace rules for common usage C implementations
Relevant Sections: 2.8.2, 2.2.3.3
Applies to 1990 edition: Yes
Related text in 1990 revised: No

Interpretation Request

Section 2.8.2 places restrictions on the symbols that can be defined by the implementation when a header is included. However, it makes no explicit statements about definition of symbols by the implementation in any other way, such as link-time resolution of a symbol with external scope, or a compiler reserving a word. An implementation with C Standard Language-Dependent Support would presumably be bound by the C Standard in this regard. However, the requirements on an implementation with Common Usage C Language-Dependent Support are not clear. Possible interpretations range from there being no such requirements, to the requirements being the same as those in the C Standard.

As concrete examples, consider an application that uses the symbols "brk," "asm," or "unix." All of these symbols are defined by traditional implementations of C on the UNIX¹ operating system via mechanisms other than inclusion of headers, and uses in an application will reference those definitions. However, an application that strictly conforms to the C Standard can define these symbols for its own use and be assured that they will not be assigned special meaning by the implementation. The requested interpretation should explain any requirements on the treatment of these symbols by an implementation with Common Usage C Language-Dependent Support. The interpretation need not treat all of these three examples the same. It should, if possible, give clear rules for resolving other examples.

IEEE Interpretation for 1003.1-1988

There is no demand in the standard for Common Usage C Language-Dependent Support implementations to avoid namespace pollution of the types mentioned in the request for interpretation. There is a requirement in 2.2.3.3 for an implementation to document the differences between the interface provided by their implementation and the interface that would have been provided had the C Standard been referenced. This should include the documentation of all cases of namespace pollution as specified in this request for interpretation.

Rationale for Interpretation

There are clearly a number of problems relating to Common Usage C addressed in IEEE Std 1003.1-1988. This causes considerable problems in producing a reasonable interpretation in these issues. The question that needs to be asked is "What can reasonably be demanded of an implementor to address these problems?" There are three fundamental options: do nothing, require documentation, and require systems to be changed to avoid the problem.

The first option is seen as unhelpful since it provides no information to the application writer and increases the effort in porting applications to the system. The third option is probably impractical since many implementors are already well advanced in generating conforming systems. Requiring this level of change would slow down the emergence of conforming systems. The second option seems the most pragmatic approach. All implementation suppliers are required to generate a conformance document and information of this type could easily be fitted into this document. The interpretation given does not deviate from the statements in

¹UNIX is a trademark of Unix System Laboratories in the US and other countries.