

Softway Systems INTERIX[™] Release 2.2
POSIX.1 Conformance Document

© 1998 Softway Systems, Inc. All rights reserved. Based upon material copyright © 1994, 1995 by Microsoft Corporation.

INTERIX is a trademark of Softway Systems, Inc.

Microsoft, MS, MS-DOS, and Windows are registered trademarks and Windows NT and the Windows logo are trademarks of Microsoft Corporation.

June 1998

Table of Contents

TABLE OF CONTENTS.....	i
PREFACE.....	iii
PURPOSE AND AUDIENCE.....	iii
MANUAL ORGANIZATION.....	iii
NOTATIONS AND CONVENTIONS	iii
SECTION 1: GENERAL.....	1
1.1 SCOPE.....	1
1.3 CONFORMANCE	1
SECTION 2: TERMINOLOGY AND GENERAL REQUIREMENTS.....	3
2.2 DEFINITIONS	3
2.3 GENERAL CONCEPTS	3
2.4 ERROR NUMBERS	3
2.5 PRIMITIVE-SYSTEM DATA TYPES.....	4
2.6 ENVIRONMENT DESCRIPTION.....	4
2.7 C LANGUAGE DEFINITIONS	4
2.8 NUMERICAL LIMITS	4
2.9 SYMBOLIC CONSTANTS.....	5
SECTION 3: PROCESS PRIMITIVES.....	6
3.1 PROCESS CREATION AND EXECUTION.....	6
3.2 PROCESS TERMINATION.....	6
3.3 SIGNALS.....	6
SECTION 4: PROCESS ENVIRONMENT	8
4.2 USER IDENTIFICATION	8
4.4 SYSTEM IDENTIFICATION	8
4.5 TIME	9
4.6 ENVIRONMENT VARIABLES.....	9
4.7 TERMINAL IDENTIFICATION	9
4.8 CONFIGURABLE SYSTEM VARIABLES.....	10
SECTION 5: DIRECTORIES AND FILES.....	11
5.1 DIRECTORIES	11
5.2 WORKING DIRECTORY	11
5.3 GENERAL FILE CREATION	12
5.4 SPECIAL FILE CREATION	12
5.5 FILE REMOVAL	13
5.6 FILE CHARACTERISTICS	13
5.7 CONFIGURABLE PATHNAME VARIABLES.....	14
SECTION 6: INPUT AND OUTPUT PRIMITIVES.....	15
6.4 INPUT AND OUTPUT	15
6.5 CONTROL OPERATIONS ON FILES	15
SECTION 7: DEVICE- AND CLASS-SPECIFIC FUNCTIONS	17
7.1 GENERAL TERMINAL INTERFACE.....	17

7.2 GENERAL TERMINAL INTERFACE CONTROL FUNCTIONS	Error! Bookmark not defined.
SECTION 8: LANGUAGE-SPECIFIC SERVICES FOR C.....	19
8.1 REFERENCED C LANGUAGE ROUTINES	19
8.2 C LANGUAGE INPUT/OUTPUT FUNCTIONS	19
8.3 OTHER C LANGUAGE FUNCTIONS.....	20
SECTION 9: SYSTEM DATABASES.....	21
9.1 SYSTEM DATABASES	21
9.2 DATABASE ACCESS	21
SECTION 10: DATA INTERCHANGE FORMAT	22
10.1 ARCHIVE/INTERCHANGE FILE FORMAT.....	22
INDEX	23

Preface

Purpose and Audience

The purpose of this document is to meet the requirements outlined in §1.3.1.2 of the POSIX.1 Standard.

This manual is intended for C programmers who are writing POSIX-conforming programs and need to know the behavior of the implementation-defined features discussed in the POSIX.1 Standard.

A conformance document with the following information shall be available for an implementation claiming conformance to this part of ISO/IEC 9945. The conformance document shall have the same structure as this part of ISO/IEC 9945, with the information presented in the appropriately numbered sections, clauses, and subclauses. The conformance document shall not contain information about extended facilities or capabilities outside the scope of this part of ISO/IEC 9945.

The conformance document shall contain a statement that indicates the full name, number, and date of the standard that applies. The conformance document may also list international software standards that are available for use by a Conforming POSIX.1 Application. Applicable characteristics where documentation is required by one of these standards, or by standards of government bodies, may also be included.

The conformance document shall describe the limit values found in `<limits.h>` and `<unistd.h>` headers, stating values, the conditions under which those values may change, and the limits of such variations, if any.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in this part of ISO/IEC 9945. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. The conformance document may specify the behavior of the implementation for those features where this part of ISO/IEC 9945 states that implementations may vary or where features are identified as undefined or unspecified.

Manual Organization

This manual follows the POSIX.1 Standard in format and structure. Section 1 contains the POSIX conformance statement, the C Standard conformance statement, and the contents of the `<limits.h>` and `<unistd.h>` header files. Sections 2 through 10 correspond to the same sections in the POSIX.1 Standard.

Notations and Conventions

Throughout this manual, the following typographic notations are used:

- **Fixed-width** font (Courier) is used to reference C language header files, utility names, and to provide examples of system input/output.
- *Italic* typeface is used to reference symbolic function parameters, C language function names, C language data types, and global external variables. Examples: `printf()`, `argc`, `uid_t`, and `errno`.
- **Bold** typeface is used to reference environment variables and with the term **NULL** pointer.
- Braces are used to enclose symbolic constants and limits defined by various header files. Examples: `{ARG_MAX}` and `{CLK_TCK}`.
- Brackets are used to enclose symbolic error codes, which are set by many of the functions. Examples: `[ENOENT]` and `[ENOMEM]`.

Section 1: General

1.1 Scope

This document defines how INTERIX™ Release 2.2 for Windows NT™ implements the implementation-defined and selected unspecified features in the ISO/IEC 9945-1:1990 POSIX Standard; henceforth, known as the POSIX.1 Standard. (This standard is entirely equivalent to IEEE 1003.1-1990.)

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

For an application to run with the behavior specified by the standard, Windows NT systems must be configured to deny the “Bypass Traverse Check” privilege to those users running POSIX.1 applications. Denying the Bypass Traverse Check privilege means that when access to a file is requested, the user must have execute permission on all of the directory components of the path. If the Bypass Traverse Check privilege is granted, access is checked only on the last component of the path.

To deny the Bypass Traverse Check privilege, complete the following steps.

1. Log in as Administrator.
2. Start the User Manager.
3. From the Policies menu, select User Rights.
4. Choose the Show Advanced User Rights button.
5. In the Bypass Traverse Checking area, replace “Everyone” with a list of the users who don’t require POSIX.1 conformance.

The system must be configured so that all files and directories, which are to be used by POSIX applications, have ACLs that will allow the *stat()* and *fstat()* functions to return correct permission bits. The simplest way to do this is to use the *chmod* program. For example, you would type:

```
chmod -r mode pathname
```

It is also possible to use the File Manager’s Security Editor to ensure that the ACLs on the files in a hierarchy will generate correct POSIX permission bits. For details, see Chapter 4, “File Manager,” in the *Microsoft Windows NT System Guide*.

1.3.1.2 Documentation

INTERIX Release 2.2 for Microsoft Windows NT conforms to ISO/IEC 9945-1:1990 (IEEE Std 1003.1-1990), Information Technology – Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language].

1.3.3 Language-Dependent Services for the C Programming Language

1.3.3.2 C Standard Language-Dependent System Support

INTERIX Release 2.2 meets the requirements of POSIX.1, Section 8 by reference to ISO/IEC 9989: 1990, Information Technology – Programming Languages – C/ANSI X3.189-1989, Programming Language – C.

Section 2: Terminology and General Requirements

2.2 Definitions

2.2.2 General Terms

2.2.2.4 Appropriate Privileges

A process cannot acquire appropriate privileges.

2.2.2.9 Character Special Files

INTERIX Release 2.2 does not provide any character special files.

2.2.2.55 Parent Process ID

When the lifetime of the parent process of a child process ends, the parent process ID of the child process is assigned to a special system process ID of 1.

2.2.2.57 Pathname

Pathnames, which begin with two consecutive slashes, are interpreted as identifying a drive specifier. The format is `//x`, where `x` is a disk drive available under Windows NT. For example, `//A/tmp` would identify the file `/tmp` on drive A.

2.2.2.68 Process Lifetime

When a process terminates, its thread of control and address space are returned to the system.

2.2.2.69 Read-Only File System

INTERIX Release 2.2 does not provide a mechanism to mark an NTFS file system as read-only. NTFS is the only file system type supported by Windows NT that provides all the required semantics of POSIX.1.

2.2.2.83 Supplementary Group ID

A process's effective group ID is included in its list of supplementary group IDs.

2.3 General Concepts

2.3.1 Extended Security Controls

INTERIX Release 2.2 does not implement any extended security controls.

2.3.2 File Access Permissions

Windows NT further provides access control lists (ACL) in NTFS as an additional mechanism for controlling file access. ACLs cannot be manipulated via the POSIX subsystem, but an ACL can further restrict access to a file beyond its permission bits.

2.4 Error Numbers

The error [EFAULT] is reliably detected.

The maximum size of a file is 2^{32} bytes in length (4,294,967,296 bytes).

2.5 Primitive-System Data Types

These type symbols are defined in the indicated headers. The symbols are not specified by POSIX.1.

Type Symbol	Defined In
<i>time_t</i>	<sys/types.h>
<i>wchar_t</i>	<ctype.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>
<i>wctype_t</i>	<ctype.h>, <stdio.h>

2.6 Environment Description

Environment variable names may consist of characters from the portable filename character set, as well as other 8-bit characters, excluding equals ('=') and the NUL character ('\0').

2.7 C Language Definitions

2.7.2 POSIX.1 Symbols

The `{_POSIX_SOURCE}` feature test macro is supported.

2.8 Numerical Limits

2.8.3 Run-Time Increasable Values

Symbolic Constant	Value
NGROUPS_MAX	16

2.8.4 Run-Time Invariant Values

Symbolic Constant	Value
ARG_MAX	14500
CHILD_MAX	195
OPEN_MAX	32
STREAM_MAX	20
TZNAME_MAX	10

2.8.5 Pathname Variable Values

Symbolic Constant	Value
LINK_MAX	1024
MAX_CANON	255
MAX_INPUT	255
NAME_MAX	255
PATH_MAX	512
PIPE_BUF	512

2.9 Symbolic Constants

2.9.3 Compile-Time Symbolic Constants for Portability Specifications

The information shown in the following table is from the `<unistd.h>` header file.

Symbolic Variable	Value	Comments
_POSIX_JOB_CONTROL		is defined
_POSIX_SAVED_IDS		is defined

2.9.4 Execution-Time Symbolic Constants for Portability Specifications

The information shown in the following table is from the `<unistd.h>` header file.

Symbolic Variable	Value	Comments
_POSIX_CHOWN_RESTRICTED	1	
_POSIX_NO_TRUNC	1	
_POSIX_VDISABLE	0	

The constant `{_POSIX_CHOWN_RESTRICTED}` is defined in the `<unistd.h>` header file with a value of 1 and the option is provided on all files.

The constant `{_POSIX_NO_TRUNC}` is defined in the `<unistd.h>` header file with a value of 1 and the option is provided on all files.

The constant `{_POSIX_VDISABLE}` is defined in the `<unistd.h>` header files with a value of 0 and the option is provided on all files.

Section 3: Process Primitives

3.1 Process Creation and Execution

3.1.1 Process Creation

3.1.1.2 Description

The calling process (parent) and the new process (child), after a call to the *fork()* function, cannot share open directory streams.

3.1.1.4 Errors

The *fork()* function does not support the detection of the [ENOMEM] error code.

3.1.2 Execute a File

3.1.2.2 Description

When the environment variable **PATH** is not defined, the search for the executable file when calling the *execlp()* and *execvp()* functions is limited to the current directory.

When an *exec* function fails to execute, but is able to locate a process image file, the *st_atime* field for the process image file is not updated.

3.1.2.4 Errors

The execution of files other than regular files is not supported.

The *exec* functions support the detection of the [ENOMEM] error code.

3.2 Process Termination

3.2.2 Terminate a Process

3.2.2.2 Description

The child processes of a terminated process are assigned a new parent process ID of 1.

3.3 Signals

3.3.1 Signal Concepts

3.3.1.2 Signal Generation and Delivery

A signal is discarded upon generation when the signal is being blocked and the action associated with the signal is SIG_IGN.

When a pending signal is generated again, the signal is only delivered once.

There are no conditions, other than those specified by this part of ISO/IEC 9945, under which the implementation generates signals.

3.3.1.3 Signal Actions

When a process sets the actions for the SIGCHLD signal to SIG_IGN, the signal will be ignored.

A SIGCHLD signal will not be generated when a process establishes a signal-catching function for the SIGCHLD signal while it has a terminated child process for which it has not waited.

3.3.3 Manipulate Signal Sets

3.3.3.4 Errors

If the value of the *signo* argument is an unsupported signal number, then the [EINVAL] error condition is detected by the *sigaddset()*, *sigdelset()*, and *sigismember()* functions.

3.3.6 Examine Pending Signals

3.3.6.4 Errors

The *sigpending()* function detects the [EFAULT] error code when the value of the *set* argument is an invalid pointer.

Section 4: Process Environment

4.2 User Identification

4.2.2 Set User and Group IDs

4.2.2.2 Description

There are no methods for obtaining appropriate privileges to change the real user ID, effective user ID, saved set-user-ID, real group ID, effective group ID, or saved set-group-ID.

4.2.3 Get Supplementary Group IDs

4.2.3.2 Description

The values of the array entries in *grouplist[]* (with indices larger than or equal to the returned value from the *getgroups()* function) are unchanged by a call to the *getgroups()* function.

4.2.4 Get User Name

4.2.4.3 Returns

The login name indicated by the pointer returned from a call to the *getlogin()* function is overwritten on each call to the *getlogin()* function.

4.2.4.4 Errors

The *getlogin()* function does not support the detection of errors.

4.4 System Identification

4.4.1 Get System Name

4.4.1.2 Description

The values shown in the following table are supported for each member of the *utsname* structure:

Member Name	Values	Comments
<i>sysname</i>	Windows NT	Operating system name
<i>nodename</i>		Name of node within network
<i>release</i>	3.51 or 4.0	Current release of Windows NT host
<i>version</i>	SPn	Current service pack of Windows NT host
<i>machine</i>	i386 i486 Pentium Intel Unknown R4000 Alpha 21064 Alpha 21164 Alpha Unknown PowerPC 601 PowerPC 603 PowerPC 604 PowerPC 603+ PowerPC 604+ PowerPC 620 PowerPC Unknown	Hardware platform

4.4.1.4 Errors

The *uname()* function returns the [EFAULT] error code when the *name* argument is an invalid address.

4.5 Time**4.5.1 Get System Time****4.5.1.4 Errors**

The *time()* function sets *errno* to the [EFAULT] error code when the *tloc* argument is an invalid address. In addition, the *time()* function returns -1 when the date is not within the range 1970 to 2105.

4.5.2 Get Process Times**4.5.2.3 Returns**

The return value from a call to the *times()* function can overflow the range of type *clock_t*.

4.5.2.4 Errors

The *times()* function returns -1 and sets *errno* to the [EFAULT] error code when the *buffer* argument is an invalid address.

4.6 Environment Variables**4.6.1 Environment Access****4.6.1.3 Returns**

The pointer returned from the *getenv()* function does not address static data; therefore, it is not overwritten on each call to the *getenv()* function.

4.6.1.4 Errors

The *getenv()* function does not support the detection of errors.

4.7 Terminal Identification**4.7.1 General Terminal Pathname****4.7.1.3 Returns**

The pathname addressed by the pointer returned from a call to the *ctermid()* function is overwritten on each call to the *ctermid()* function.

4.7.1.4 Errors

The *ctermid()* function does not support the detection of errors.

4.7.2 Determine Terminal Device Name**4.7.2.2 Description**

The pathname addressed by the pointer returned from a call to the *ttyname()* function is overwritten on each call to the *ttyname()* function.

4.7.2.4 Errors

The *ttyname()* and *isatty()* functions do not support the detection of errors.

4.8 Configurable System Variables

4.8.1 Get Configurable System Variables

4.8.1.2 Description

Only those *name* values specified in Table 4-2 of ISO/IEC 9945-1:1990 are supported by the *sysconf()* function.

Section 5: Directories and Files

5.1 Directories

5.1.2 Directory Operations

5.1.2.2 Description

The *readdir()* function does not return entries for dot or dot-dot.

When a file is added to or removed from a directory after a call to the *opendir()* or *rewinddir()* functions, the change is reflected in the set of files returned via the *readdir()* function call.

The link count of a directory is not incremented when a subdirectory is created.

A directory pointer *dirp* is unusable after a call to the *closedir()* function.

5.1.2.4 Errors

The *opendir()* function supports the detection of the [EMFILE] and [ENFILE] error codes under the conditions described in the POSIX.1 Standard.

The *readdir()*, *closedir()*, and *rewinddir()* functions support the detection of the [EBADF] error code when *dirp* does not refer to an open directory stream.

5.2 Working Directory

5.2.2 Get Working Directory Pathname

5.2.2.2 Description

When a call to the *getcwd()* function is made and the first argument (*buf*) is **NULL**, the system returns **NULL** and sets *errno* to the [EFAULT] error code.

5.2.2.3 Returns

The contents of the buffer passed to the *getcwd()* function after an error has occurred are unchanged.

5.2.2.4 Errors

The *getcwd()* function does not support the detection of the [EACCES] error code.

5.3 General File Creation

5.3.1 Open a File

5.3.1.2 Description

When a call to the *open()* function is made with **O_CREAT** set, a new file's group is set to the group ID of its parent directory.

When a call to the *open()* function is made with **O_CREAT** set, the bits in the third argument (other than the mode bits) are ignored.

When a call to the *open()* function is made with **O_EXCL** set and **O_CREAT** not set, the **O_EXCL** flag is ignored.

When a call to the *open()* function is made for file types other than socket, FIFO, block special and character special, the status of the `O_NONBLOCK` flag is ignored.

When a call to the *open()* function is made for file types other than FIFO special files, regular files, and terminal device files, the value of the `O_TRUNC` flag is ignored.

When a call to the *open()* function is made with `O_TRUNC` and `O_RDONLY` set, the value of the `O_TRUNC` flag is ignored.

5.3.3 Set File Creation Mask

5.3.3.2 Description

Only the permission bits are supported; therefore, the meaning of the bits (other than the permission bits) in the argument to the *umask()* function are ignored.

5.3.4 Link to a File

5.3.4.2 Description

Linking across file systems is not supported.

Linking to a directory is not supported; therefore, obtaining appropriate privileges to link to a directory is not supported.

When calling the *link()* function, the calling process needs permission to access the existing file.

5.4 Special File Creation

5.4.1 Make a Directory

5.4.1.2 Description

When calling the *mkdir()* function, the bits (other than the permission bits in `mode`) have no effect.

When calling the *mkdir()* function, a new directory's group is set to the group ID of its parent directory.

5.4.2 Make a FIFO Special File

5.4.2.2 Description

When calling the *mkfifo()* function, the bits (other than the permission bits in `mode`) have no effect.

When calling the *mkfifo()* function, a new FIFO group is set to the group ID of its parent directory.

5.5 File Removal

5.5.1 Remove Directory Entries

5.5.1.2 Description

The use of the *unlink()* function on directories is not supported; therefore, obtaining the appropriate privileges to use the *unlink()* function on a directory is not supported.

5.5.1.4 Errors

Calling the *unlink()* function with a directory is not supported; therefore, the *unlink()* function does not set *errno* to the `[EBUSY]` error code and return `-1` when the directory named by the *path* argument cannot be unlinked because it is being used by the system or another process.

5.5.2 Remove a Directory

5.5.2.2 Description

A call to the *rmdir()* function will fail if the named directory is the root directory.

Calling the *rmdir()* function, when the named directory is the root directory or the current working directory of any process, does not succeed and sets *errno* to the [EBUSY] error code.

5.5.2.4 Errors

A call to the *rmdir()* function does not fail if the named directory is being used by another process.

5.5.3 Rename a File

5.5.3.2 Description

Write access is not required for the *old* directory or *new* directory (if it exists) in order for the call to the *rename(old,new)* function to be successful.

5.5.3.4 Errors

A call to *rename(old,new)* does not fail when the named directory is being used by another process.

5.6 File Characteristics

5.6.1 File Characteristics: Header and Data Structure

The *st_size* field in the *stat* structure for FIFO files contains its PIPE_BUF value.

5.6.2 Get File Status

5.6.2.2 Description

No additional or alternate access control mechanisms are defined which would cause *stat()* or *fstat()* to fail.

5.6.3 Check File Accessibility

5.6.3.4 Errors

The *access()* function supports the detection of the [EINVAL] error code under the conditions described in the POSIX.1 Standard.

5.6.4 Change File Modes

5.6.4.2 Description

Calling the *chmod()* function on a file with open file descriptors has no effect on the open file descriptors.

INTERIX Release 2.2 restricts the *chmod()* function such that no user has permission to set the {S_ISUID} or {S_ISGID} bits on any file; these bits are always ignored.

5.6.5 Change Owner and Group of a File

5.6.5.4 Errors

The *chown()* function supports the detection of the [EINVAL] error code under the conditions described in the POSIX.1 Standard.

5.6.6 Set File Access and Modification Times

5.6.6.2 Description

INTERIX Release 2.2 does not extend the *utimbuf* structure.

5.7 Configurable Pathname Variables

5.7.1 Get Configurable Pathname Variables

5.7.1.2 Description

Only the name variables listed in Table 5-2 of ISO/IEC 9945-1:1990 are supported.

When calling the *pathconf()* or *fpathconf()* functions, the association of the variable name `{_PC_PIPE_BUF}` is supported for all file types.

5.7.1.4 Errors

The *pathconf()* function supports the detection of the `[EACCES]`, `[ENAMETOOLONG]`, `[ENOENT]`, `[ENOTDIR]`, and `[EINVAL]` error codes under the conditions described in the POSIX.1 Standard.

The *fpathconf()* function supports the detection of the `[EINVAL]` and `[EBADF]` error codes under the conditions described in the POSIX.1 Standard.

Section 6: Input and Output Primitives

6.4 Input and Output

6.4.1 Read from a File

6.4.1.2 Description

When the *read()* function is interrupted by a signal after it has successfully read some data, it returns the number of bytes read.

Device special files are not supported; therefore, the *read()* function is not supported when the starting position is at or after the end-of-file for device special files.

When the *nbyte* parameter exceeds *SSIZE_MAX*, the return value from the *read()* function is truncated to type *ssize_t*.

6.4.1.4 Errors

When the device reports a hardware error, the *read()* function returns *-1* and sets *errno* to the [EIO] error code.

6.4.2 Write to a File

6.4.2.2 Description

When calling the *write()* function on a file that is not a regular file with *nbyte* set to zero, the function returns *0*.

When the *write()* function is interrupted by a signal after it successfully writes some data, it returns the number of bytes written.

When the *nbyte* parameter exceeds *SSIZE_MAX*, the return value from the *write()* function is truncated to type *ssize_t*.

6.4.2.2 Errors

When the device reports a hardware error, the *write()* function returns *-1* and sets *errno* to the [EIO] error code.

6.5 Control Operations on Files

6.5.2 File Control

6.5.2.2 Description

Advisory record locking is supported only for regular files.

If *l_len* is negative, then the lock is placed on the file starting with offset *l_start - l_len* through *l_start - 1*. If the starting point is before the beginning of the file, then *fcntl()* returns *-1* and sets *errno* to the [EINVAL] error code.

6.5.2.2 Errors

The *fcntl()* function does not support the detection of the [EDEADLK] error code.

6.5.3 Reposition Read/Write File Offset

6.5.3.2 Description

Pipes and FIFOs are incapable of seeking; therefore, a call to the *lseek()* function returns -1 and sets *errno* to the [ESPIPE] error code when the *fildev* argument is associated with a file of these types.

Section 7: Device- and Class-Specific Functions

7.1 General Terminal Interface

INTERIX Release 2.2 does not provide asynchronous communication ports (serial communications). Most terminal semantics have been implemented for windowed terminals and pseudo terminals.

7.1.1 Interface Characteristics

7.1.1.3 The Controlling Terminal

If a session leader with no controlling terminal opens a terminal not already associated with a session and the `O_NOCTTY` is clear, then this terminal will become the controlling terminal for this session. If all file descriptors to a controlling terminal by a session are closed then this terminal remains the controlling terminal until all members of the session have exited or have created a new session.

7.1.1.5 Input Processing and Reading Data

When `{MAX_INPUT}` is exceeded the behavior is determined by the setting of the `IMAXBEL` bit with `c_iflag`. If `IMAXBEL` is set then no more input will be accepted and the terminal will sound its bell. If `IMAXBEL` is clear then input will continue to be accepted after the input and output queues have been flushed.

7.1.1.10 Modem Disconnect

For INTERIX Release 2.2 `CLOCAL` is always treated as clear.

7.1.2 Parameters that Can Be Set

7.1.2.2 Input Modes

The break condition is only defined for asynchronous communication ports.

The initial control value for `c_iflag` is with the following bits set: `BRKINT`, `ICRCL`, `IMAXBEL` and `IXON`.

7.1.2.3 Output Modes

The initial control value for `c_oflag` is with the following bits set: `OPOST` and `ONLCR`.

7.1.2.4 Control Modes

The initial control value for `c_cflag` is with the following bits set: `CREAD` and `CS8`.

7.1.2.5 Local Modes

The initial control value for `c_lflag` is with the following bits set: `ICANON`, `ECHO`, `ECHOE`, `ECHOK` and `ISIG`.

7.1.2.6 Special Control Characters

The initial control value for `c_cc` members are the following values:

Member Name	Values
VEOF	<code>^D</code>
VEOL	<code>{_POSIX_VDISABLE}</code>
VERASE	<code>^H</code>
VINTR	<code>^C</code>

VKILL	^U
VQUIT	^\
VSUSP	^Z
VSTART	^Q
VSTOP	^S
VMIN	1
VTIME	0
VBELTIME	3

7.2 General Terminal Interface Control Functions

7.2.1 Get and Set State

7.2.1.2 Description

For INTERIX Release 2.2 changes in baud rate apply only to asynchronous communication ports.

7.2.2 Line Control Functions

7.2.2.2 Description

For INTERIX Release 2.2 these function apply only to asynchronous communication ports.

Section 8: Language-Specific Services for C

8.1 Referenced C Language Routines

8.1.1 Extensions to Time Functions

If the `TZ` variable is of the “:characters” format, the characters following the colon are ignored.

8.1.2 Extensions to the `setlocale()` Function

8.1.2.2 Description

The *category* argument defines the functions affected by the `setlocale()` function. The *locale* argument is a pointer to a string that specified the name of the locale.

When the *locale* argument points to an empty string, environment variables can provide a value (`$LANG`, `$LC_ALL`, or specific variables for the category being set). If none of these environment variables is present or if they have `NULL` values, the *locale* argument defaults to the value `C` (the minimal ANSI conforming environment for C translation).

The *locale* argument can be either `C` or `POSIX`.

8.2 C Language Input/Output Functions

8.2.1 Map a Stream Pointer to a File Descriptor

8.2.1.4 Errors

The `fileno()` function does not support the detection of errors.

8.2.2 Open a Stream on a File Descriptor

8.2.2.2 Description

There are no additional values supported for the *type* argument beyond those specified in the POSIX.1 Standard.

8.2.2.4 Errors

The `fdopen()` function supports the detection of errors. If the *type* argument is not one of those specified in the POSIX.1 Standard, a call to the `fdopen()` function returns `-1` and sets `errno` to the `[EINVAL]` error code.

8.2.3 Interactions of Other FILE-Type C Functions

Output may be seen twice if a process forks while data is buffered for a stream. Similarly, input may be seen twice under the same circumstances. Otherwise, output and input will be seen exactly once.

8.3 Other C Language Functions

8.3.2 Set Time Zone

8.3.2.2 Description

When the **TZ** variable is absent from the environment, the system default value of “PST8PDT” is used for the time-zone.

Section 9: System Databases

9.1 System Databases

The initial working directory may be set from the Windows NT User Manager. If the initial working directory is **NULL**, then a value of “//C/” is used. The initial working program is always “/bin/ksh”.

Only the fields specified in the POSIX.1 Standard for the *group* and *user* databases are available to a POSIX-compliant program.

9.2 Database Access

9.2.1 Group Database Access

9.2.1.3 Returns

The data returned from calls to the functions *getgrgid()* and *getgrnam()* overwrites the data from previous calls to either function. For example, a call to the *getgrgid()* function overwrites data from a previous call to the *getgrnam()* function.

9.2.1.4 Errors

The *getgrgid()* and *getgrnam()* functions do not support the detection of errors.

9.2.2 User Database Access

9.2.2.3 Returns

The data returned from calls to the functions *getpwuid()* and *getpwnam()* overwrites the data from previous calls to either function. For example, a call to the *getpwuid()* function overwrites data from a previous call to the *getpwnam()* function.

9.2.2.4 Errors

The *getpwuid()* and *getpwnam()* functions do not support the detection of errors.

Section 10: Data Interchange Format

10.1 Archive/Interchange File Format

The name of the *format-creating* utility and *format-reading* utility is `pax`. The `pax` utility is fully described in the IEEE POSIX.2-1992 Standard.

10.1.1 Extended tar Format

The `pax` utility supports the use of 8-bit characters in names for users and groups; the character encoding associated with the current locale is used when displaying these names.

When data is found on the medium that would create an invalid file name, the `pax` utility does not create the file on the file hierarchy and displays an error message to show that the entry is not being stored.

10.1.2 Extended cpio Format

10.1.2.1 cpio Header

The `c_rdev` field is always set to zero.

10.1.2.2 cpio File Name

When data is found on the medium that would create an invalid file name, the `pax` utility does not create the file on the file hierarchy and displays an error message to show that the entry is not being stored.

10.1.3 Multiple Volumes

The `pax` format-creating utility for the `tar` and `cpio` formats determine what file to read or write for the next volume of a multi-volume archive by prompting the user.

Index

—\$—
\$LANG, 19
\$LC_ALL, 19

—_—
_POSIX_CHOWN_RESTRICTED, 5
_POSIX_JOB_CONTROL, 5
_POSIX_NO_TRUNC, 5
_POSIX_SAVED_IDS, 5
_POSIX_SOURCE, 4
_POSIX_VDISABLE, 5
_POSIX_VERSION, 5

—A—
access(), 14
ARG_MAX, 4

—B—
buf, 11
buffer, 9

—C—
c_rdev, 22
category, 19
cfgetispeed(), 18
cfgetospeed(), 18
cfsetispeed(), 18
cfsetospeed(), 18
CHILD_MAX, 4
chmod, 1
chmod(), 14
chown(), 14
clock_t, 9
closedir(), 11
cpio, 22
ctermid(), 10

—D—
dirp, 11

—E—
EACCES, 11, 13, 14
EBADF, 11, 14
EBUSY, 13

EDEADLK, 15
EFAULT, 3, 7, 9, 11
EINVAL, 7, 14, 15, 19
EIO, 15
EMFILE, 11
ENAMETOOLONG, 14
ENFILE, 11
ENOENT, 14
ENOMEM, 6
ENOTDIR, 14
errno, 9, 11, 13, 15, 16, 19
ESPIPE, 16
exec, 6
execvp(), 6
execvp(), 6

—F—
fcntl(), 15
fdopen(), 19
FIFO, 13
fileno(), 19
fork(), 6
fpathconf(), 14
fstat(), 1

—G—
getcwd(), 11
getenv(), 10
getgrgid(), 21
getgrnam(), 21
getgroups(), 8
getlogin(), 8
getpwnam(), 21
getpwuid(), 21
grouplist, 8

—I—
ICANON, 17
IEXTEN, 17
isatty(), 10
ISIG, 17
IXOFF, 17
IXON, 17

—L—
l_len, 15
l_start, 15
link(), 12

LINK_MAX, 4
 locale, 19
 lseek(), 16

—M—

machine, 9
 MAX_CANON, 4
 MAX_INPUT, 4
 mkdir(), 12
 mkfifo(), 12, 13

—N—

name, 9, 10
 NAME_MAX, 4
 nbyte, 15
 NGROUPS_MAX, 4
 nodename, 9
 noshell, 21
 NULL, 11, 19, 21

—O—

O_CREAT, 11, 12
 O_EXCL, 12
 O_NONBLOCK, 12
 O_RDONLY, 12
 O_TRUNC, 12
 open(), 11, 12, 17, 18
 OPEN_MAX, 4
 opendir(), 11
 OPOST, 17

—P—

PATH, 6, 13
 PATH_MAX, 4
 pathconf(), 14
 pax, 22
 PIPE files, 16
 PIPE_BUF, 4, 13, 14

—R—

read(), 15
 readdir(), 11
 release, 9
 rename(), 13
 rewinddir(), 11
 rmdir(), 13

—S—

S_ISGID, 14

S_ISUID, 14
 set, 7
 setlocale(), 19
 SIG_IGN, 6, 7
 sigaddset(), 7
 SIGCHLD, 7
 sigdelset(), 7
 sigismember(), 7
 signo, 7
 sigpending(), 7
 SSIZE_MAX, 4, 15
 ssize_t, 15
 st_atime, 6
 st_size, 13
 START character, 17
 stat, 13
 stat(), 1
 STOP character, 17
 STREAM_MAX, 4
 sysconf(), 10
 sysname, 9

—T—

tar, 22
 tcsendbreak(), 18
 tcsetattr(), 18
 time(), 9
 time_t, 4
 times(), 9
 time-zone, 20
 tloc, 9
 ttyname(), 10
 type, 19
 TZ, 19, 20
 TZNAME_MAX, 4

—U—

umask(), 12
 uname(), 9
 unlink(), 13
 utimebuf, 14

—V—

version, 9

—W—

wchar_t, 4
 wctype_t, 4
 write(), 15, 17