

# **Linux Standard Base Core Specification for IA64 3.1**

## **Linux Standard Base Core Specification for IA64 3.1**

Copyright © 2004, 2005 Free Standards Group

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Portions of the text are copyrighted by the following parties:

- The Regents of the University of California
- Free Software Foundation
- Ian F. Darwin
- Paul Vixie
- BSDI (now Wind River)
- Andrew G Morgan
- Jean-loup Gailly and Mark Adler
- Massachusetts Institute of Technology

These excerpts are being used in accordance with their respective licenses.

Linux is a trademark of Linus Torvalds.

UNIX a registered trademark of the Open Group in the United States and other countries.

LSB is a trademark of the Free Standards Group in the USA and other countries.

AMD is a trademark of Advanced Micro Devices, Inc.

Intel and Itanium are registered trademarks and Intel386 is a trademarks of Intel Corporation.

PowerPC and PowerPC Architecture are trademarks of the IBM Corporation.

OpenGL is a registered trademark of Silicon Graphics, Inc.

# Contents

<b>Foreword</b> .....	<b>vi</b>
<b>Introduction</b> .....	<b>vii</b>
<b>I Introductory Elements</b> .....	<b>8</b>
1 Scope.....	9
1.1 General.....	9
1.2 Module Specific Scope.....	9
2 References.....	10
2.1 Normative References .....	10
2.2 Informative References/Bibliography .....	12
3 Requirements .....	14
3.1 Relevant Libraries .....	14
3.2 LSB Implementation Conformance .....	14
3.3 LSB Application Conformance.....	15
4 Definitions .....	17
5 Terminology .....	18
6 Documentation Conventions .....	20
<b>II Executable and Linking Format (ELF)</b> .....	<b>21</b>
7 Introduction.....	22
8 Low Level System Information.....	23
8.1 Machine Interface.....	23
8.2 Function Calling Sequence.....	27
8.3 Operating System Interface .....	28
8.4 Process Initialization.....	29
8.5 Coding Examples .....	31
8.6 C Stack Frame .....	32
8.7 Debug Information.....	33
9 Object Format .....	34
9.1 Introduction .....	34
9.2 ELF Header .....	34
9.3 Sections.....	35
9.4 Symbol Table .....	37
9.5 Relocation.....	37
10 Program Loading and Dynamic Linking .....	38
10.1 Introduction .....	38
10.2 Program Header.....	38
10.3 Program Loading .....	38
10.4 Dynamic Linking.....	38
<b>III Base Libraries</b> .....	<b>40</b>
11 Libraries .....	41
11.1 Program Interpreter/Dynamic Linker .....	41
11.2 Interfaces for libc .....	41
11.3 Data Definitions for libc .....	55
11.4 Interfaces for libm .....	80
11.5 Data Definitions for libm.....	84
11.6 Interfaces for libpthread.....	90
11.7 Data Definitions for libpthread .....	93
11.8 Interfaces for libgcc_s .....	97
11.9 Data Definitions for libgcc_s.....	98
11.10 Interface Definitions for libgcc_s.....	101

11.11 Interfaces for libdl .....	105
11.12 Data Definitions for libdl .....	106
11.13 Interfaces for libcrypt.....	106
<b>IV Utility Libraries.....</b>	<b>108</b>
12 Libraries .....	109
12.1 Interfaces for libz.....	109
12.2 Data Definitions for libz .....	109
12.3 Interfaces for libncurses.....	110
12.4 Data Definitions for libncurses.....	110
12.5 Interfaces for libutil.....	116
<b>V Package Format and Installation .....</b>	<b>117</b>
13 Software Installation .....	118
13.1 Package Dependencies .....	118
13.2 Package Architecture Considerations .....	118
<b>A Alphabetical Listing of Interfaces.....</b>	<b>119</b>
A.1 libgcc_s.....	119
A.2 libm.....	119
<b>B GNU Free Documentation License (Informative) .....</b>	<b>120</b>
B.1 PREAMBLE .....	120
B.2 APPLICABILITY AND DEFINITIONS.....	120
B.3 VERBATIM COPYING .....	121
B.4 COPYING IN QUANTITY .....	121
B.5 MODIFICATIONS .....	122
B.6 COMBINING DOCUMENTS.....	123
B.7 COLLECTIONS OF DOCUMENTS.....	124
B.8 AGGREGATION WITH INDEPENDENT WORKS.....	124
B.9 TRANSLATION .....	124
B.10 TERMINATION .....	124
B.11 FUTURE REVISIONS OF THIS LICENSE .....	125
B.12 How to use this License for your documents.....	125

## List of Figures

8-1 Structure Smaller Than A Word .....	25
8-2 No Padding.....	25
8-3 Internal and Tail Padding.....	26
8-4 Bit-Field Ranges .....	27

## Foreword

1            This is version 3.1 of the Linux Standard Base Core Specification for IA64. This  
2            specification is part of a family of specifications under the general title "Linux  
3            Standard Base". Developers of applications or implementations interested in using  
4            the LSB trademark should see the Free Standards Group Certification Policy for  
5            details.

## Introduction

1           The LSB defines a binary interface for application programs that are compiled and  
2 packaged for LSB-conforming implementations on many different hardware  
3 architectures. Since a binary specification shall include information specific to the  
4 computer processor architecture for which it is intended, it is not possible for a  
5 single document to specify the interface for all possible LSB-conforming  
6 implementations. Therefore, the LSB is a family of specifications, rather than a single  
7 one.

8           This document should be used in conjunction with the documents it references. This  
9 document enumerates the system components it includes, but descriptions of those  
10 components may be included entirely or partly in this document, partly in other  
11 documents, or entirely in other reference documents. For example, the section that  
12 describes system service routines includes a list of the system routines supported in  
13 this interface, formal declarations of the data structures they use that are visible to  
14 applications, and a pointer to the underlying referenced specification for  
15 information about the syntax and semantics of each call. Only those routines not  
16 described in standards referenced by this document, or extensions to those  
17 standards, are described in the detail. Information referenced in this way is as much  
18 a part of this document as is the information explicitly included here.

19           The specification carries a version number of either the form  $x.y$  or  $x.y.z$ . This  
20 version number carries the following meaning:

- 21           • The first number ( $x$ ) is the major version number. All versions with the same  
22           major version number should share binary compatibility. Any addition or  
23           deletion of a new library results in a new version number. Interfaces marked as  
24           *deprecated* may be removed from the specification at a major version change.
- 25           • The second number ( $y$ ) is the minor version number. Individual interfaces may be  
26           added if all certified implementations already had that (previously  
27           undocumented) interface. Interfaces may be marked as *deprecated* at a minor  
28           version change. Other minor changes may be permitted at the discretion of the  
29           LSB workgroup.
- 30           • The third number ( $z$ ), if present, is the editorial level. Only editorial changes  
31           should be included in such versions.

32           Since this specification is a descriptive Application Binary Interface, and not a source  
33 level API specification, it is not possible to make a guarantee of 100% backward  
34 compatibility between major releases. However, it is the intent that those parts of the  
35 binary interface that are visible in the source level API will remain backward  
36 compatible from version to version, except where a feature marked as "Deprecated"  
37 in one release may be removed from a future release.

38           Implementors are strongly encouraged to make use of symbol versioning to permit  
39 simultaneous support of applications conforming to different releases of this  
40 specification.

# I Introductory Elements



# 1 Scope

## 1.1 General

1           The Linux Standard Base (LSB) defines a system interface for compiled applications  
2           and a minimal environment for support of installation scripts. Its purpose is to  
3           enable a uniform industry standard environment for high-volume applications  
4           conforming to the LSB.

5           These specifications are composed of two basic parts: A common specification  
6           ("LSB-generic" or "generic LSB") describing those parts of the interface that remain  
7           constant across all implementations of the LSB, and an architecture-specific  
8           supplement ("LSB-arch" or "archLSB") describing the parts of the interface that vary  
9           by processor architecture. Together, the LSB-generic and the architecture-specific  
10          supplement for a single hardware architecture provide a complete interface  
11          specification for compiled application programs on systems that share a common  
12          hardware architecture.

13          The LSB-generic document shall be used in conjunction with an architecture-specific  
14          supplement. Whenever a section of the LSB-generic specification shall be  
15          supplemented by architecture-specific information, the LSB-generic document  
16          includes a reference to the architecture supplement. Architecture supplements may  
17          also contain additional information that is not referenced in the LSB-generic  
18          document.

19          The LSB contains both a set of Application Program Interfaces (APIs) and  
20          Application Binary Interfaces (ABIs). APIs may appear in the source code of portable  
21          applications, while the compiled binary of that application may use the larger set of  
22          ABIs. A conforming implementation shall provide all of the ABIs listed here. The  
23          compilation system may replace (e.g. by macro definition) certain APIs with calls to  
24          one or more of the underlying binary interfaces, and may insert calls to binary  
25          interfaces as needed.

26          The LSB is primarily a binary interface definition. Not all of the source level APIs  
27          available to applications may be contained in this specification.

## 1.2 Module Specific Scope

28          This is the Itanium architecture specific Core module of the Linux Standards Base  
29          (LSB). This module supplements the generic LSB Core module with those interfaces  
30          that differ between architectures.

31          Interfaces described in this module are mandatory except where explicitly listed  
32          otherwise. Core interfaces may be supplemented by other modules; all modules are  
33          built upon the core.

## 2 References

### 2.1 Normative References

1 The following referenced documents are indispensable for the application of this  
2 document. For dated references, only the edition cited applies. For undated  
3 references, the latest edition of the referenced document (including any  
4 amendments) applies.

5 **Note:** Where copies of a document are available on the World Wide Web, a Uniform  
6 Resource Locator (URL) is given for informative purposes only. This may point to a more  
7 recent copy of the referenced specification, or may be out of date. Reference copies of  
8 specifications at the revision level indicated may be found at the Free Standards Group's  
9 Reference Specifications (<http://refspecs.freestandards.org>) site.

10 **Table 2-1 Normative References**

Name	Title	URL
Filesystem Hierarchy Standard	Filesystem Hierarchy Standard (FHS) 2.3	<a href="http://www.pathname.com/fhs/">http://www.pathname.com/fhs/</a>
IEC 60559/IEEE 754 Floating Point	IEC 60559:1989 Binary floating-point arithmetic for microprocessor systems	<a href="http://www.ieee.org/">http://www.ieee.org/</a>
Intel® Itanium™ Processor-specific Application Binary Interface	Intel® Itanium™ Processor-specific Application Binary Interface	<a href="http://refspecs.freestandards.org/elf/IA64-SysV-psABI.pdf">http://refspecs.freestandards.org/elf/IA64-SysV-psABI.pdf</a>
ISO C (1999)	ISO/IEC 9899: 1999, Programming Languages --C	
ISO POSIX (2003)	ISO/IEC 9945-1:2003 Information technology -- Portable Operating System Interface (POSIX) -- Part 1: Base Definitions ISO/IEC 9945-2:2003 Information technology -- Portable Operating System Interface (POSIX) -- Part 2: System Interfaces ISO/IEC 9945-3:2003 Information technology -- Portable Operating System Interface (POSIX) -- Part 3: Shell and Utilities ISO/IEC 9945-4:2003	<a href="http://www.unix.org/version3/">http://www.unix.org/version3/</a>

Name	Title	URL
	Information technology -- Portable Operating System Interface (POSIX) -- Part 4: Rationale  Including Technical Cor. 1: 2004	
Itanium™ Architecture Software Developer's Manual Volume 1	Itanium™ Architecture Software Developer's Manual Volume 1: Application Architecture	<a href="http://refspecs.freestandards.org/IA64-softdevman-vol1.pdf">http://refspecs.freestandards.org/IA64-softdevman-vol1.pdf</a>
Itanium™ Architecture Software Developer's Manual Volume 2	Itanium™ Architecture Software Developer's Manual Volume 2: System Architecture	<a href="http://refspecs.freestandards.org/IA64-softdevman-vol2.pdf">http://refspecs.freestandards.org/IA64-softdevman-vol2.pdf</a>
Itanium™ Architecture Software Developer's Manual Volume 3	Itanium™ Architecture Software Developer's Manual Volume 3: Instruction Set Reference	<a href="http://refspecs.freestandards.org/IA64-softdevman-vol3.pdf">http://refspecs.freestandards.org/IA64-softdevman-vol3.pdf</a>
Itanium™ Architecture Software Developer's Manual Volume 4	IA-64 Processor Reference: Intel® Itanium™ Processor Reference Manual for Software Development	<a href="http://refspecs.freestandards.org/IA64-softdevman-vol4.pdf">http://refspecs.freestandards.org/IA64-softdevman-vol4.pdf</a>
Itanium™ Software Conventions and Runtime Guide	Itanium™ Software Conventions & Runtime Architecture Guide, September 2000	<a href="http://refspecs.freestandards.org/IA64conventions.pdf">http://refspecs.freestandards.org/IA64conventions.pdf</a>
Large File Support	Large File Support	<a href="http://www.UNIX-systems.org/version2/whatsnew/lfs20mar.html">http://www.UNIX-systems.org/version2/whatsnew/lfs20mar.html</a>
SUSv2	CAE Specification, January 1997, System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)	<a href="http://www.opengroup.org/publications/catalog/un.htm">http://www.opengroup.org/publications/catalog/un.htm</a>
SUSv2 Commands and Utilities	The Single UNIX® Specification(SUS) Version 2, Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)	<a href="http://www.opengroup.org/publications/catalog/un.htm">http://www.opengroup.org/publications/catalog/un.htm</a>
SVID Issue 3	American Telephone and Telegraph Company, System V Interface Definition, Issue 3 ;	

Name	Title	URL
	Morristown, NJ, UNIX Press, 1989.(ISBN 0201566524)	
SVID Issue 4	System V Interface Definition,Fourth Edition	
System V ABI	System V Application Binary Interface, Edition 4.1	<a href="http://www.caldera.com/developers/devspecs/gabi41.pdf">http://www.caldera.com/developers/devspecs/gabi41.pdf</a>
System V ABI Update	System V Application Binary Interface - DRAFT - 17 December 2003	<a href="http://www.caldera.com/developers/gabi/2003-12-17/contents.html">http://www.caldera.com/developers/gabi/2003-12-17/contents.html</a>
X/Open Curses	CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), plus Corrigendum U018	<a href="http://www.opengroup.org/publications/catalog/un.htm">http://www.opengroup.org/publications/catalog/un.htm</a>

11

## 2.2 Informative References/Bibliography

12

In addition, the specifications listed below provide essential background information to implementors of this specification. These references are included for information only.

13

14

15

**Table 2-2 Other References**

Name	Title	URL
DWARF Debugging Information Format, Revision 2.0.0	DWARF Debugging Information Format, Revision 2.0.0 (July 27, 1993)	<a href="http://refspecs.freestandards.org/dwarf/dwarf-2.0.0.pdf">http://refspecs.freestandards.org/dwarf/dwarf-2.0.0.pdf</a>
DWARF Debugging Information Format, Revision 3.0.0 (Draft)	DWARF Debugging Information Format, Revision 3.0.0 (Draft)	<a href="http://refspecs.freestandards.org/dwarf/">http://refspecs.freestandards.org/dwarf/</a>
ISO/IEC TR14652	ISO/IEC Technical Report 14652:2002 Specification method for cultural conventions	
ITU-T V.42	International Telecommunication Union Recommendation V.42 (2002): Error-correcting procedures for DCEs using asynchronous-to-synchronous conversion ITUV	<a href="http://www.itu.int/rec/recommendation.asp?type=folders&amp;lang=e&amp;parent=T-REC-V.42">http://www.itu.int/rec/recommendation.asp?type=folders&amp;lang=e&amp;parent=T-REC-V.42</a>
Li18nux Globalization	LI18NUNIX 2000	<a href="http://www.li18nux.org">http://www.li18nux.org</a>

<b>Name</b>	<b>Title</b>	<b>URL</b>
Specification	Globalization Specification, Version 1.0 with Amendment 4	/docs/html/LI18NWX-2000-amd4.htm
Linux Allocated Device Registry	LINUX ALLOCATED DEVICES	<a href="http://www.lanana.org/docs/device-list/devices.txt">http://www.lanana.org/docs/device-list/devices.txt</a>
PAM	Open Software Foundation, Request For Comments: 86.0, October 1995, V. Samar & R.Schemers (SunSoft)	<a href="http://www.opengroup.org/tech/rfc/mirror-rfc/rfc86.0.txt">http://www.opengroup.org/tech/rfc/mirror-rfc/rfc86.0.txt</a>
RFC 1321: The MD5 Message-Digest Algorithm	IETF RFC 1321: The MD5 Message-Digest Algorithm	<a href="http://www.ietf.org/rfc/rfc1321.txt">http://www.ietf.org/rfc/rfc1321.txt</a>
RFC 1831/1832 RPC & XDR	IETF RFC 1831 & 1832	<a href="http://www.ietf.org/">http://www.ietf.org/</a>
RFC 1833: Binding Protocols for ONC RPC Version 2	IETF RFC 1833: Binding Protocols for ONC RPC Version 2	<a href="http://www.ietf.org/rfc/rfc1833.txt">http://www.ietf.org/rfc/rfc1833.txt</a>
RFC 1950: ZLIB Compressed Data Format Specification	IETF RFC 1950: ZLIB Compressed Data Format Specification	<a href="http://www.ietf.org/rfc/rfc1950.txt">http://www.ietf.org/rfc/rfc1950.txt</a>
RFC 1951: DEFLATE Compressed Data Format Specification	IETF RFC 1951: DEFLATE Compressed Data Format Specification version 1.3	<a href="http://www.ietf.org/rfc/rfc1951.txt">http://www.ietf.org/rfc/rfc1951.txt</a>
RFC 1952: GZIP File Format Specification	IETF RFC 1952: GZIP file format specification version 4.3	<a href="http://www.ietf.org/rfc/rfc1952.txt">http://www.ietf.org/rfc/rfc1952.txt</a>
RFC 2440: OpenPGP Message Format	IETF RFC 2440: OpenPGP Message Format	<a href="http://www.ietf.org/rfc/rfc2440.txt">http://www.ietf.org/rfc/rfc2440.txt</a>
RFC 2821: Simple Mail Transfer Protocol	IETF RFC 2821: Simple Mail Transfer Protocol	<a href="http://www.ietf.org/rfc/rfc2821.txt">http://www.ietf.org/rfc/rfc2821.txt</a>
RFC 2822: Internet Message Format	IETF RFC 2822: Internet Message Format	<a href="http://www.ietf.org/rfc/rfc2822.txt">http://www.ietf.org/rfc/rfc2822.txt</a>
RFC 791: Internet Protocol	IETF RFC 791: Internet Protocol Specification	<a href="http://www.ietf.org/rfc/rfc791.txt">http://www.ietf.org/rfc/rfc791.txt</a>
RPM Package Format	RPM Package Format V3.0	<a href="http://www.rpm.org/max-rpm/s1-rpm-file-format-rpm-file-format.html">http://www.rpm.org/max-rpm/s1-rpm-file-format-rpm-file-format.html</a>
zlib Manual	zlib 1.2 Manual	<a href="http://www.gzip.org/zlib/">http://www.gzip.org/zlib/</a>

## 3 Requirements

### 3.1 Relevant Libraries

1           The libraries listed in Table 3-1 shall be available on IA64 Linux Standard Base  
2 systems, with the specified runtime names. These names override or supplement the  
3 names specified in the generic LSB specification. The specified program interpreter,  
4 referred to as proginterp in this table, shall be used to load the shared libraries  
5 specified by DT\_NEEDED entries at run time.

6           **Table 3-1 Standard Library Names**

Library	Runtime Name
libm	libm.so.6.1
libdl	libdl.so.2
libcrypt	libcrypt.so.1
libz	libz.so.1
libncurses	libncurses.so.5
libutil	libutil.so.1
libc	libc.so.6.1
libpthread	libpthread.so.0
proginterp	/lib/ld-lsb-ia64.so.3
libgcc_s	libgcc_s.so.1

7  
8           These libraries will be in an implementation-defined directory which the dynamic  
9 linker shall search by default.

### 3.2 LSB Implementation Conformance

10           A conforming implementation is necessarily architecture specific, and must provide  
11 the interfaces specified by both the generic LSB Core specification and its relevant  
12 architecture specific supplement.

13           **Rationale:** An implementation must provide *at least* the interfaces specified in these  
14 specifications. It may also provide additional interfaces.

15           A conforming implementation shall satisfy the following requirements:

- 16           • A processor architecture represents a family of related processors which may not  
17           have identical feature sets. The architecture specific supplement to this  
18           specification for a given target processor architecture describes a minimum  
19           acceptable processor. The implementation shall provide all features of this  
20           processor, whether in hardware or through emulation transparent to the  
21           application.
- 22           • The implementation shall be capable of executing compiled applications having  
23           the format and using the system interfaces described in this document.
- 24           • The implementation shall provide libraries containing the interfaces specified by  
25           this document, and shall provide a dynamic linking mechanism that allows these

- 26 interfaces to be attached to applications at runtime. All the interfaces shall behave  
27 as specified in this document.
- 28 • The map of virtual memory provided by the implementation shall conform to the  
29 requirements of this document.
  - 30 • The implementation's low-level behavior with respect to function call linkage,  
31 system traps, signals, and other such activities shall conform to the formats  
32 described in this document.
  - 33 • The implementation shall provide all of the mandatory interfaces in their entirety.
  - 34 • The implementation may provide one or more of the optional interfaces. Each  
35 optional interface that is provided shall be provided in its entirety. The product  
36 documentation shall state which optional interfaces are provided.
  - 37 • The implementation shall provide all files and utilities specified as part of this  
38 document in the format defined here and in other referenced documents. All  
39 commands and utilities shall behave as required by this document. The  
40 implementation shall also provide all mandatory components of an application's  
41 runtime environment that are included or referenced in this document.
  - 42 • The implementation, when provided with standard data formats and values at a  
43 named interface, shall provide the behavior defined for those values and data  
44 formats at that interface. However, a conforming implementation may consist of  
45 components which are separately packaged and/or sold. For example, a vendor of  
46 a conforming implementation might sell the hardware, operating system, and  
47 windowing system as separately packaged items.
  - 48 • The implementation may provide additional interfaces with different names. It  
49 may also provide additional behavior corresponding to data values outside the  
50 standard ranges, for standard named interfaces.

### 3.3 LSB Application Conformance

51 A conforming application is necessarily architecture specific, and must conform to  
52 both the generic LSB Core specification and its relevant architecture specific  
53 supplement.

54 A conforming application shall satisfy the following requirements:

- 55 • Its executable files shall be either shell scripts or object files in the format defined  
56 for the Object File Format system interface.
- 57 • Its object files shall participate in dynamic linking as defined in the Program  
58 Loading and Linking System interface.
- 59 • It shall employ only the instructions, traps, and other low-level facilities defined in  
60 the Low-Level System interface as being for use by applications.
- 61 • If it requires any optional interface defined in this document in order to be  
62 installed or to execute successfully, the requirement for that optional interface  
63 shall be stated in the application's documentation.
- 64 • It shall not use any interface or data format that is not required to be provided by a  
65 conforming implementation, unless:
  - 66 • If such an interface or data format is supplied by another application through  
67 direct invocation of that application during execution, that application shall be  
68 in turn an LSB conforming application.

### 3 Requirements

69                   • The use of that interface or data format, as well as its source, shall be identified  
70                   in the documentation of the application.

71                   • It shall not use any values for a named interface that are reserved for vendor  
72                   extensions.

73                   A strictly conforming application shall not require or use any interface, facility, or  
74                   implementation-defined extension that is not defined in this document in order to be  
75                   installed or to execute successfully.



## 4 Definitions

1	For the purposes of this document, the following definitions, as specified in the
2	<i>ISO/IEC Directives, Part 2, 2001, 4th Edition</i> , apply:
3	can
4	be able to; there is a possibility of; it is possible to
5	cannot
6	be unable to; there is no possibility of; it is not possible to
7	may
8	is permitted; is allowed; is permissible
9	need not
10	it is not required that; no...is required
11	shall
12	is to; is required to; it is required that; has to; only...is permitted; it is necessary
13	shall not
14	is not allowed [permitted] [acceptable] [permissible]; is required to be not; is
15	required that...be not; is not to be
16	should
17	it is recommended that; ought to
18	should not
19	it is not recommended that; ought not to

## 5 Terminology

1 For the purposes of this document, the following terms apply:

2 archLSB

3 The architectural part of the LSB Specification which describes the specific parts  
4 of the interface that are platform specific. The archLSB is complementary to the  
5 gLSB.

6 Binary Standard

7 The total set of interfaces that are available to be used in the compiled binary  
8 code of a conforming application.

9 gLSB

10 The common part of the LSB Specification that describes those parts of the  
11 interface that remain constant across all hardware implementations of the LSB.

12 implementation-defined

13 Describes a value or behavior that is not defined by this document but is  
14 selected by an implementor. The value or behavior may vary among  
15 implementations that conform to this document. An application should not rely  
16 on the existence of the value or behavior. An application that relies on such a  
17 value or behavior cannot be assured to be portable across conforming  
18 implementations. The implementor shall document such a value or behavior so  
19 that it can be used correctly by an application.

20 Shell Script

21 A file that is read by an interpreter (e.g., awk). The first line of the shell script  
22 includes a reference to its interpreter binary.

23 Source Standard

24 The set of interfaces that are available to be used in the source code of a  
25 conforming application.

26 undefined

27 Describes the nature of a value or behavior not defined by this document which  
28 results from use of an invalid program construct or invalid data input. The  
29 value or behavior may vary among implementations that conform to this  
30 document. An application should not rely on the existence or validity of the  
31 value or behavior. An application that relies on any particular value or behavior  
32 cannot be assured to be portable across conforming implementations.

33 unspecified

34 Describes the nature of a value or behavior not specified by this document  
35 which results from use of a valid program construct or valid data input. The  
36 value or behavior may vary among implementations that conform to this  
37 document. An application should not rely on the existence or validity of the  
38 value or behavior. An application that relies on any particular value or behavior  
39 cannot be assured to be portable across conforming implementations.

40 Other terms and definitions used in this document shall have the same meaning as  
41 defined in Chapter 3 of the Base Definitions volume of ISO POSIX (2003).

## 6 Documentation Conventions

1 Throughout this document, the following typographic conventions are used:

2 `function()`

3 the name of a function

4 **command**

5 the name of a command or utility

6 `CONSTANT`

7 a constant value

8 *parameter*

9 a parameter

10 `variable`

11 a variable

12 Throughout this specification, several tables of interfaces are presented. Each entry  
13 in these tables has the following format:

14 `name`

15 the name of the interface

16 `(symver)`

17 An optional symbol version identifier, if required.

18 `[refno]`

19 A reference number indexing the table of referenced specifications that follows  
20 this table.

21 For example,

22 `forkpty(GLIBC_2.0) [SUSv3]`

23 refers to the interface named `forkpty()` with symbol version `GLIBC_2.0` that is  
24 defined in the `SUSv3` reference.

25 **Note:** Symbol versions are defined in the architecture specific supplements only.

## **II Executable and Linking Format (ELF)**

## 7 Introduction

1 Executable and Linking Format (ELF) defines the object format for compiled  
2 applications. This specification supplements the information found in System V ABI  
3 Update and Intel® Itanium™ Processor-specific Application Binary Interface, and is  
4 intended to document additions made since the publication of that document.

## 8 Low Level System Information

### 8.1 Machine Interface

#### 8.1.1 Processor Architecture

1 The Itanium™ Architecture is specified by the following documents

- 2 • Itanium™ Architecture Software Developer's Manual Volume 1
- 3 • Itanium™ Architecture Software Developer's Manual Volume 2
- 4 • Itanium™ Architecture Software Developer's Manual Volume 3
- 5 • Itanium™ Architecture Software Developer's Manual Volume 4
- 6 • Itanium™ Software Conventions and Runtime Guide
- 7 • Intel® Itanium™ Processor-specific Application Binary Interface

8 Only the features of the Itanium™ processor instruction set may be assumed to be  
9 present. An application should determine if any additional instruction set features  
10 are available before using those additional features. If a feature is not present, then  
11 the application may not use it.

12 Conforming applications may use only instructions which do not require elevated  
13 privileges.

14 Conforming applications shall not invoke the implementations underlying system  
15 call interface directly. The interfaces in the implementation base libraries shall be  
16 used instead.

17 **Rationale:** Implementation-supplied base libraries may use the system call interface but  
18 applications must not assume any particular operating system or kernel version is  
19 present.

20 There are some features of the Itanium™ processor architecture that need not be  
21 supported by a conforming implementation. These are described in this chapter. A  
22 conforming application shall not rely on these features.

23 Applications conforming to this specification must provide feedback to the user if a  
24 feature that is required for correct execution of the application is not present.

25 Applications conforming to this specification should attempt to execute in a  
26 diminished capacity if a required feature is not present.

27 This specification does not provide any performance guarantees of a conforming  
28 system. A system conforming to this specification may be implemented in either  
29 hardware or software.

30 This specification describes only LP64 (i.e. 32-bit integers, 64-bit longs and pointers)  
31 based implementations. Implementations may also provide ILP32 (32-bit integers,  
32 longs, and pointers), but conforming applications shall not rely on support for ILP32.  
33 See section 1.2 of the Intel® Itanium™ Processor-specific Application Binary  
34 Interface for further information.

#### 8.1.2 Data Representation

35 The following sections, in conjunction with section 4 of Itanium™ Software  
36 Conventions and Runtime Guide, define the size, alignment requirements, and  
37 hardware representation of the standard C data types.

38 Within this specification, the term `byte` refers to an 8-bit object, the term `halfword`  
 39 refers to a 16-bit object, the term `word` refers to a 32-bit object, the term `doubleword`  
 40 refers to a 64-bit object, and the term `quadword` refers to a 128-bit object.

### 41 8.1.2.1 Byte Ordering

42 LSB-conforming applications shall use little-endian byte ordering. LSB-conforming  
 43 implementations may support big-endian applications.

### 44 8.1.2.2 Fundamental Types

45 Table 8-1 describes how fundamental C language data types shall be represented:

46 **Table 8-1 Scalar Types**

Type	C	sizeof	Alignment (bytes)	Hardware Representation
Integral	<code>_Bool</code>	1	1	byte (sign unspecified)
	<code>char</code>	1	1	signed byte
	<code>signed char</code>			
	<code>unsigned char</code>			signed byte
	<code>short</code>	2	2	signed half-word
	<code>signed short</code>			
	<code>unsigned short</code>			unsigned halfword
	<code>int</code>	4	4	signed word
	<code>signed int</code>			
	<code>unsigned int</code>			unsigned word
	<code>long</code>	8	8	signed doubleword
	<code>signed long</code>			
	<code>unsigned long</code>			unsigned doubleword
	<code>long long</code>	8	8	signed doubleword
	<code>signed long long</code>			
<code>unsigned long long</code>			unsigned doubleword	
Pointer	<i>any-type</i> *	8	8	unsigned doubleword



Type	C	sizeof	Alignment (bytes)	Hardware Representation
	<i>any-type</i> (*)()			
Floating-Point	float	4	4	IEEE Single-precision
	double	8	8	IEEE Double-precision
	long double	16	16	IEEE Double-extended

47

48

A null pointer (for all types) shall have the value zero.

49

### 8.1.2.3 Aggregates and Unions

50

51

52

53

54

Aggregates (structures and arrays) and unions assume the alignment of their most strictly aligned component. The size of any object, including aggregates and unions, shall always be a multiple of the object's alignment. An array uses the same alignment as its elements. Structure and union objects may require padding to meet size and element constraints. The contents of such padding is undefined.

55

56

- An entire structure or union object shall be aligned on the same boundary as its most strictly aligned member.

57

58

- Each member shall be assigned to the lowest available offset with the appropriate alignment. This may require *internal padding*, depending on the previous member.

59

60

- A structure's size shall be increased, if necessary, to make it a multiple of the alignment. This may require *tail padding*, depending on the last member.

61

A conforming application shall not read padding.

<pre> struct {     char c; } </pre>	
Byte aligned, sizeof is 1	
Offset	Byte 0
0	$c^0$

62

63

64

Figure 8-1 Structure Smaller Than A Word

<pre> struct {     char c;     char d;     short s;     int i;     long l; } </pre>	
Doubleword Aligned, sizeof is 16	

65

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0	s <sup>2</sup>		d <sup>1</sup>	c <sup>0</sup>
4	i <sup>0</sup>			
8	l <sup>0</sup>			
12				

66

67

Figure 8-2 No Padding

<pre> struct {     char c;     long l;     int i;     short s; } </pre>				
Doubleword Aligned, sizeof is 24				
Offset	Byte 3	Byte 2	Byte 1	Byte 0
0	pad <sup>1</sup>			c <sup>0</sup>
4	pad <sup>1</sup>			
8	l <sup>0</sup>			
12				
16	i <sup>0</sup>			
20	pad <sup>2</sup>		s <sup>0</sup>	

68

69

70

Figure 8-3 Internal and Tail Padding

71

#### 8.1.2.4 Bit Fields

72

C `struct` and `union` definitions may have *bit-fields*, which define integral objects with a specified number of bits.

73

74

Bit fields that are declared with neither `signed` nor `unsigned` specifier shall always be treated as `unsigned`. Bit fields obey the same size and alignment rules as other structure and union members, with the following additional properties:

75

76

77

- Bit-fields are allocated from right to left (least to most significant).
- A bit-field must entirely reside in a storage unit for its appropriate type. A bit field shall never cross its unit boundary.
- Bit-fields may share a storage unit with other `struct/union` members, including members that are not bit fields. Such other `struct/union` members shall occupy different parts of the storage unit.
- The type of unnamed bit-fields shall not affect the alignment of a structure or union, although individual bit-field member offsets shall obey the alignment constraints.

78

79

80

81

82

83

84

85

Bit-field Type	Width <i>w</i>	Range
----------------	----------------	-------

Bit-field Type	Width $w$	Range
signed char char unsigned char	1 to 8	$-2^{w-1}$ to $2^{w-1}-1$ 0 to $2^w-1$ 0 to $2^w-1$
signed short short unsigned short	1 to 16	$-2^{w-1}$ to $2^{w-1}-1$ 0 to $2^w-1$ 0 to $2^w-1$
signed int int unsigned int	1 to 32	$-2^{w-1}$ to $2^{w-1}-1$ 0 to $2^w-1$ 0 to $2^w-1$
signed long long unsigned long	1 to 64	$-2^{w-1}$ to $2^{w-1}-1$ 0 to $2^w-1$ 0 to $2^w-1$

86

87

Figure 8-4 Bit-Field Ranges

## 8.2 Function Calling Sequence

88

89

90

LSB-conforming applications shall use the procedure linkage and function calling sequence as defined in Chapter 8.4 of the Itanium™ Software Conventions and Runtime Guide.

### 8.2.1 Registers

91

92

The CPU general and other registers are as defined in the Itanium™ Architecture Software Developer's Manual Volume 1 Section 3.1.

### 8.2.2 Floating Point Registers

93

94

The floating point registers are as defined in the Itanium™ Architecture Software Developer's Manual Volume 1 Section 3.1.

### 8.2.3 Stack Frame

95

96

The stackframe layout is as described in the Itanium™ Software Conventions and Runtime Guide Chapter 8.4.

### 8.2.4 Arguments

97

#### 8.2.4.1 Introduction

98

99

100

The procedure parameter passing mechanism is as described in the Itanium™ Software Conventions and Runtime Guide Chapter 8.5. The following subsections provide additional information.

101

#### 8.2.4.2 Integral/Pointer

102

See Itanium™ Software Conventions and Runtime Guide Chapter 8.5.

103

#### 8.2.4.3 Floating Point

104

See Itanium™ Software Conventions and Runtime Guide Chapter 8.5.

105           **8.2.4.4 Struct and Union Point**

106           See Itanium™ Software Conventions and Runtime Guide Chapter 8.5.

107           **8.2.4.5 Variable Arguments**

108           See Itanium™ Software Conventions and Runtime Guide Chapter 8.5.4.

**8.2.5 Return Values**

109           **8.2.5.1 Introduction**

110           Values are returned from functions as described in Itanium™ Software Conventions  
111           and Runtime Guide Chapter 8.6, and as further described here.

112           **8.2.5.2 Void**

113           Functions that return no value (void functions) are not required to put any  
114           particular value in any general register.

115           **8.2.5.3 Integral/Pointer**

116           See Itanium™ Software Conventions and Runtime Guide Chapter 8.6.

117           **8.2.5.4 Floating Point**

118           See Itanium™ Software Conventions and Runtime Guide Chapter 8.6.

119           **8.2.5.5 Struct and Union**

120           See Itanium™ Software Conventions and Runtime Guide Chapter 8.6 (aggregate  
121           return values). Depending on the size (including any padding), aggregate data types  
122           may be passed in one or more general registers, or in memory.

**8.3 Operating System Interface**

123           LSB-conforming applications shall use the Operating System Interfaces as defined in  
124           Chapter 3 of the Intel® Itanium™ Processor-specific Application Binary Interface.

**8.3.1 Processor Execution Mode**

125           Applications must assume that they will execute in the least privileged user mode  
126           (i.e. level 3). Other privilege levels are reserved for the Operating System.

**8.3.2 Exception Interface**

127           **8.3.2.1 Introduction**

128           LSB-conforming implementations shall support the exception interface as specified  
129           in Intel® Itanium™ Processor-specific Application Binary Interface, section 3.3.1.

130           **8.3.2.2 Hardware Exception Types**

131           See Intel® Itanium™ Processor-specific Application Binary Interface, section 3.3.1.

132           **8.3.2.3 Software Trap Types**

133           See Intel® Itanium™ Processor-specific Application Binary Interface, section 3.3.1.

### 8.3.3 Signal Delivery

134 LSB-conforming systems shall deliver signals as specified in Intel® Itanium™  
135 Processor-specific Application Binary Interface, section 3.3.2.

#### 8.3.3.1 Signal Handler Interface

136  
137 The signal handler interface shall be as specified in Intel® Itanium™  
138 Processor-specific Application Binary Interface, section 3.3.3.

### 8.3.4 Debugging Support

139 The LSB does not specify debugging information.

### 8.3.5 Process Startup

140 LSB-conforming systems shall initialize processes as specified in Intel® Itanium™  
141 Processor-specific Application Binary Interface, section 3.3.5.

## 8.4 Process Initialization

142 LSB-conforming applications shall use the Process Startup as defined in Section 3.3.5  
143 of the Intel® Itanium™ Processor-specific Application Binary Interface.

### 8.4.1 Special Registers

144 Intel® Itanium™ Processor-specific Application Binary Interface, section 3.3.5,  
145 defines required register initializations for process startup.

### 8.4.2 Process Stack (on entry)

146 As defined in Intel® Itanium™ Processor-specific Application Binary Interface,  
147 section 3.3.5, the return pointer register (rp) shall contain a valid return address,  
148 such that if the application program returns from the main entry routine, the  
149 implementation shall cause the application to exit normally, using the returned  
150 value as the exit status. Further, the unwind information for this "bottom of stack"  
151 routine in the implementation shall provide a mechanism for recognizing the bottom  
152 of the stack during a stack unwind.

### 8.4.3 Auxiliary Vector

153 The auxiliary vector conveys information from the operating system to the  
154 application. Only the terminating null auxiliary vector entry is required, but if any  
155 other entries are present, they shall be interpreted as follows. This vector is an array  
156 of the following structures.

```
157 typedef struct
158 {
159     long int a_type;           /* Entry type */
160     union
161     {
162         long int a_val;       /* Integer value */
163         void *a_ptr;         /* Pointer value */
164         void (*a_fcn) (void); /* Function pointer value */
165     } a_un;
166 } auxv_t;
```

167 The application shall interpret the a\_un value according to the a\_type. Other  
168 auxiliary vector types are reserved.

169           The `a_type` field shall contain one of the following values:  
170           AT\_NULL  
171           The last entry in the array has type AT\_NULL. The value in `a_un` is undefined.  
172           AT\_IGNORE  
173           The value in `a_un` is undefined, and should be ignored.  
174           AT\_EXECD  
175           File descriptor of program  
176           AT\_PHDR  
177           Program headers for program  
178           AT\_PHENT  
179           Size of program header entry  
180           AT\_PHNUM  
181           Number of program headers  
182           AT\_PAGESZ  
183           System page size  
184           AT\_BASE  
185           Base address of interpreter  
186           AT\_FLAGS  
187           Flags  
188           AT\_ENTRY  
189           Entry point of program  
190           AT\_NOTELF  
191           Program is not ELF  
192           AT\_UID  
193           Real uid  
194           AT\_EUID  
195           Effective uid  
196           AT\_GID  
197           Real gid  
198           AT\_EGID  
199           Effective gid  
200           AT\_CLKTCK  
201           Frequency of `times()`

202	AT_PLATFORM
203	String identifying platform.
204	AT_HWCAP
205	Machine dependent hints about processor capabilities.
206	AT_FPUCW
207	Used FPU control word
208	AT_DCACHEBSIZE
209	Data cache block size
210	AT_ICACHEBSIZE
211	Instruction cache block size
212	AT_UCACHEBSIZE
213	Unified cache block size
214	<b>Note:</b> The auxiliary vector is intended for passing information from the operating
215	system to the program interpreter.

#### 8.4.4 Environment

216 Although a pointer to the environment vector should be available as a third  
 217 argument to the `main()` entry point, conforming applications should use `getenv()`  
 218 to access the environment. (See ISO POSIX (2003), Section `exec()`).

## 8.5 Coding Examples

### 8.5.1 Introduction

219 LSB-conforming applications may implement fundamental operations using the  
 220 Coding Examples as shown below.  
 221 Sample code sequences and coding conventions can be found in Itanium™ Software  
 222 Conventions and Runtime Guide, Chapter 9.

### 8.5.2 Code Model Overview/Architecture Constraints

223 As defined in Intel® Itanium™ Processor-specific Application Binary Interface,  
 224 relocatable files, executable files, and shared object files that are supplied as part of  
 225 an application shall use Position Independent Code, as described in Itanium™  
 226 Software Conventions and Runtime Guide, Chapter 12.

### 8.5.3 Position-Independent Function Prologue

227 See Itanium™ Software Conventions and Runtime Guide, Chapter 8.4.

### 8.5.4 Data Objects

228 See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.3.4,  
 229 and Itanium™ Software Conventions and Runtime Guide, Chapter 12.3.

#### 8.5.4.1 Absolute Load & Store

230 Conforming applications shall not use absolute addressing.  
 231

232 **8.5.4.2 Position Relative Load & Store**

233 See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.3.4.

**8.5.5 Function Calls**

234 See Itanium™ Software Conventions and Runtime Guide, Chapter 8.4.

235 Four types of procedure call are defined in Itanium™ Software Conventions and  
236 Runtime Guide, Chapter 8.3. Although special calling conventions are permitted,  
237 provided that the compiler and runtime library agree on these conventions, none are  
238 defined for this standard. Consequently, no application shall depend on a type of  
239 procedure call other than Direct Calls, Direct Dynamically Linked Calls, or Indirect  
240 Calls, as defined in Itanium™ Software Conventions and Runtime Guide, Chapter  
241 8.3.

242 **8.5.5.1 Absolute Direct Function Call**

243 Conforming applications shall not use absolute addressing.

244 **8.5.5.2 Absolute Indirect Function Call**

245 Conforming applications shall not use absolute addressing.

246 **8.5.5.3 Position-Independent Direct Function Call**

247 See Itanium™ Software Conventions and Runtime Guide, Chapter 8.4.1.

248 **8.5.5.4 Position-Independent Indirect Function Call**

249 See Itanium™ Software Conventions and Runtime Guide, Chapter 8.4.2.

**8.5.6 Branching**

250 Branching is described in Itanium™ Architecture Software Developer's Manual  
251 Volume 4, Chapter 4.5.

252 **8.5.6.1 Branch Instruction**

253 See Itanium™ Architecture Software Developer's Manual Volume 4, Chapter 4.5.

254 **8.5.6.2 Absolute switch() code**

255 Conforming applications shall not use absolute addressing.

256 **8.5.6.3 Position-Independent switch() code**

257 Where there are several possible targets for a branch, the compiler may use a  
258 number of different code generation strategies. See Itanium™ Software  
259 Conventions and Runtime Guide, Chapter 9.1.7.

**8.6 C Stack Frame**

**8.6.1 Variable Argument List**

260 See Itanium™ Software Conventions and Runtime Guide, Chapter 8.5.2, and 8.5.4.

**8.6.2 Dynamic Allocation of Stack Space**

261 The C library `alloca()` function should be used to dynamically allocate stack space.



## 8.7 Debug Information

262

The LSB does not currently specify the format of Debug information.

# 9 Object Format

## 9.1 Introduction

1            LSB-conforming implementations shall support an object file , called Executable and  
2            Linking Format (ELF) as defined by the System V ABI, Intel® Itanium™  
3            Processor-specific Application Binary Interface and as supplemented by the Linux  
4            Standard Base Specification and this document.

## 9.2 ELF Header

### 9.2.1 Machine Information

5            LSB-conforming applications shall use the Machine Information as defined in Intel®  
6            Itanium™ Processor-specific Application Binary Interface, Chapter 4.  
7            Implementations shall support the LP64 model. It is unspecified whether or not the  
8            ILP32 model shall also be supported.

#### 9.2.1.1 File Class

9  
10           For LP64 relocatable objects, the file class value in `e_ident[EI_CLASS]` may be  
11           either `ELFCLASS32` or `ELFCLASS64`, and a conforming linker must be able to  
12           process either or both classes.

#### 9.2.1.2 Data Encoding

13  
14           Implementations shall support 2's complement, little endian data encoding. The data  
15           encoding value in `e_ident[EI_DATA]` shall contain the value `ELFDATA2LSB`.

#### 9.2.1.3 OS Identification

16  
17           The OS Identification field `e_ident[EI_OSABI]` shall contain the value  
18           `ELFOSABI_NONE`.

#### 9.2.1.4 Processor Identification

19  
20           The processor identification value held in `e_machine` shall contain the value  
21           `EM_IA_64`.

#### 9.2.1.5 Processor Specific Flags

22  
23           The flags field `e_flags` shall be as described in Intel® Itanium™ Processor-specific  
24           Application Binary Interface, Chapter 4.1.1.6.

25           The following additional processor-specific flags are defined:

26           **Table 9-1 Additional Processor-Specific Flags**

Name	Value
<code>EF_IA_64_LINUX_EXECUTABLE_STACK</code>	<code>0x00000001</code>

27  
28           `EF_IA_64_LINUX_EXECUTABLE_STACK`

29           The stack and heap sections are executable. If this flag is not set, code can not be  
30           executed from the stack or heap.

## 9.3 Sections

31 The Itanium™ architecture defines two processor-specific section types, as described  
 32 in Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 4.

### 9.3.1 Special Sections

33 The following sections are defined in the Intel® Itanium™ Processor-specific  
 34 Application Binary Interface.

35 **Table 9-2 ELF Special Sections**

Name	Type	Attributes
.got	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE+SHF_IA_64_SHORT
.IA_64.archext	SHT_IA_64_EXT	0
.IA_64.pltoff	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE+SHF_IA_64_SHORT
.IA_64.unwind	SHT_IA_64_UNWIND	SHF_ALLOC+SHF_LINK_ORDER
.IA_64.unwind_info	SHT_PROGBITS	SHF_ALLOC
.plt	SHT_PROGBITS	SHF_ALLOC+SHF_EXECINSTR
.sbss	SHT_NOBITS	SHF_ALLOC+SHF_WRITE+SHF_IA_64_SHORT
.sdata	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE+SHF_IA_64_SHORT
.sdata1	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE+SHF_IA_64_SHORT

36

37 .got

38 This section holds the Global Offset Table. See 'Coding Examples' in Chapter 3,  
 39 'Special Sections' in Chapter 4, and 'Global Offset Table' in Chapter 5 of the  
 40 processor supplement for more information.

41 .IA\_64.archext

42 This section holds product-specific extension bits. The link editor will perform a  
 43 logical "or" of the extension bits of each object when creating an executable so  
 44 that it creates only a single .IA\_64.archext section in the executable.

45 .IA\_64.pltoff

46 This section holds local function descriptor entries.

47 .IA\_64.unwind

48 This section holds the unwind function table. The contents are described in the  
 49 Intel (r) Itanium (tm) Processor Specific ABI.

50            `.IA_64.unwind_info`

51            This section holds stack unwind and and exception handling information. The  
52            exception handling information is programming language specific, and is  
53            unspecified.

54            `.plt`

55            This section holds the Procedure Linkage Table.

56            `.sbss`

57            This section holds uninitialized data that contribute to the program's memory  
58            image. Data objects contained in this section are recommended to be eight bytes  
59            or less in size. The system initializes the data with zeroes when the program  
60            begins to run. The section occupies no file space, as indicated by the section type  
61            SHT\_NOBITS. The `.sbss` section is placed so it may be accessed using short  
62            direct addressing (22 bit offset from `gp`).

63            `.sdata`

64            This section and the `.sdata1` section hold initialized data that contribute to the  
65            program's memory image. Data objects contained in this section are  
66            recommended to be eight bytes or less in size. The `.sdata` and `.sdata1` sections  
67            are placed so they may be accessed using short direct addressing (22 bit offset  
68            from `gp`).

69            `.sdata1`

70            See `.sdata`.

### 9.3.2 Linux Special Sections

71            The following Linux IA-64 specific sections are defined here.

72            **Table 9-3 Additional Special Sections**

Name	Type	Attributes
<code>.opd</code>	SHT_PROGBITS	SHF_ALLOC
<code>.rela.dyn</code>	SHT_RELA	SHF_ALLOC
<code>.rela.IA_64.pltoff</code>	SHT_RELA	SHF_ALLOC

73

74            `.opd`

75            This section holds function descriptors

76            `.rela.dyn`

77            This section holds relocation information, as described in 'Relocation'. These  
78            relocations are applied to the `.dyn` section.

79            `.rela.IA_64.pltoff`

80            This section holds relocation information, as described in 'Relocation'. These  
81            relocations are applied to the `.IA_64.pltoff` section.

### 9.3.3 Section Types

82 Section Types are described in the Intel® Itanium™ Processor-specific Application  
 83 Binary Interface, Chapter 4.2. LSB conforming implementations are not required to  
 84 use any sections in the range from `SHT_IA_64_LOPSREG` to `SHT_IA_64_HIPSREG`.  
 85 Additionally, LSB conforming implementations are not required to support the  
 86 `SHT_IA_64_PRIORITY_INIT` section, beyond the gABI requirements for the handling  
 87 of unrecognized section types, linking them into a contiguous section in the object  
 88 file created by the static linker.

### 9.3.4 Section Attribute Flags

89 LSB-conforming implementations shall support the section attribute flags specified  
 90 in Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 4.2.2.

### 9.3.5 Special Section Types

91 The special section types `SHT_IA64_EXT` and `SHT_IA64_UNWIND` are defined in Intel®  
 92 Itanium™ Processor-specific Application Binary Interface, Chapter 4.2.1.

## 9.4 Symbol Table

93 If an executable file contains a reference to a function defined in one of its associated  
 94 shared objects, the symbol table section for that file shall contain an entry for that  
 95 symbol. The `st_shndx` member of that symbol table entry contains `SHN_UNDEF`. This  
 96 signals to the dynamic linker that the symbol definition for that function is not  
 97 contained in the executable file itself. If that symbol has been allocated a procedure  
 98 linkage table entry in the executable file, and the `st_value` member for that symbol  
 99 table entry is non-zero, the value shall contain the virtual address of the first  
 100 instruction of that procedure linkage table entry. Otherwise, the `st_value` member  
 101 contains zero. This procedure linkage table entry address is used by the dynamic  
 102 linker in resolving references to the address of the function.

## 9.5 Relocation

### 9.5.1 Relocation Types

103 LSB-conforming systems shall support the relocation types described in Intel®  
 104 Itanium™ Processor-specific Application Binary Interface, Chapter 4.3.

# 10 Program Loading and Dynamic Linking

## 10.1 Introduction

1           LSB-conforming implementations shall support the object file information and  
2           system actions that create running programs as specified in the System V ABI, Intel®  
3           Itanium™ Processor-specific Application Binary Interface and as supplemented by  
4           the Linux Standard Base Specification and this document.

## 10.2 Program Header

5           The program header shall be as defined in the Intel® Itanium™ Processor-specific  
6           Application Binary Interface, Chapter 5.

### 10.2.1 Types

7           See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.1.

### 10.2.2 Flags

8           See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.1.

## 10.3 Program Loading

9           See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.2.

## 10.4 Dynamic Linking

10          See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.3.

### 10.4.1 Dynamic Entries

#### 10.4.1.1 ELF Dynamic Entries

11          The following dynamic entries are defined in the Intel® Itanium™  
12          Processor-specific Application Binary Interface, Chapter 5.3.2.

13          DT\_PLTGOT

14          This entry's `d_ptr` member gives the address of the first byte in the procedure  
15          linkage table  
16          

#### 10.4.1.2 Additional Dynamic Entries

17          The following dynamic entries are defined here.

18          DT\_RELACOUNT

19          The number of relative relocations in `.rela.dyn`  
20          

### 10.4.2 Global Offset Table

21          See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.3.4.

### 10.4.3 Shared Object Dependencies

22          See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.3.3.

#### **10.4.4 Function Addresses**

23 See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.3.5.

#### **10.4.5 Procedure Linkage Table**

24 See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.3.6.

#### **10.4.6 Initialization and Termination Functions**

25 See Intel® Itanium™ Processor-specific Application Binary Interface, Chapter 5.3.7.

## **III Base Libraries**



## 11 Libraries

1 An LSB-conforming implementation shall support base libraries which provide  
2 interfaces for accessing the operating system, processor and other hardware in the  
3 system.

4 Only those interfaces that are unique to the Itanium™ platform are defined here.  
5 This section should be used in conjunction with the corresponding section in the  
6 Linux Standard Base Specification.

### 11.1 Program Interpreter/Dynamic Linker

7 The Program Interpreter shall be `/lib/ld-lsb-ia64.so.3`.

### 11.2 Interfaces for libc

8 Table 11-1 defines the library name and shared object name for the libc library

9 **Table 11-1 libc Definition**

Library:	libc
SONAME:	libc.so.6.1

10  
11 The behavior of the interfaces in this library is specified by the following specifica-  
12 tions:

[LFS] Large File Support  
[LSB] This Specification  
[SUSv2] SUSv2  
[SUSv3] ISO POSIX (2003)  
[SVID.3] SVID Issue 3  
13 [SVID.4] SVID Issue 4

#### 11.2.1 RPC

##### 11.2.1.1 Interfaces for RPC

14  
15 An LSB conforming implementation shall provide the architecture specific functions  
16 for RPC specified in Table 11-2, with the full mandatory functionality as described in  
17 the referenced underlying specification.

18 **Table 11-2 libc - RPC Function Interfaces**

authnone_create( GLIBC_2.2) [SVID.4]	clnt_create(GLIBC _2.2) [SVID.4]	clnt_pcreateerror( GLIBC_2.2) [SVID.4]	clnt_pererrno(GLIB C_2.2) [SVID.4]
clnt_perror(GLIB C_2.2) [SVID.4]	clnt_screateerror (GLIBC_2.2) [SVID.4]	clnt_spererrno(GLI BC_2.2) [SVID.4]	clnt_sperror(GLIB C_2.2) [SVID.4]
key_decryptsessio n(GLIBC_2.2) [SVID.3]	pmap_getport(GL IBC_2.2) [LSB]	pmap_set(GLIBC_ 2.2) [LSB]	pmap_unset(GLIB C_2.2) [LSB]
svc_getreqset(GLI	svc_register(GLIB	svc_run(GLIBC_2.	svc_sendreply(GL

BC_2.2) [SVID.3]	C_2.2) [LSB]	2) [LSB]	IBC_2.2) [LSB]
svcerr_auth(GLIBC_2.2) [SVID.3]	svcerr_decode(GLIBC_2.2) [SVID.3]	svcerr_noproc(GLIBC_2.2) [SVID.3]	svcerr_noprog(GLIBC_2.2) [SVID.3]
svcerr_progvers(GLIBC_2.2) [SVID.3]	svcerr_systemerr(GLIBC_2.2) [SVID.3]	svcerr_weakauth(GLIBC_2.2) [SVID.3]	svctcp_create(GLIBC_2.2) [LSB]
svcdup_create(GLIBC_2.2) [LSB]	xdr_accepted_reply(GLIBC_2.2) [SVID.3]	xdr_array(GLIBC_2.2) [SVID.3]	xdr_bool(GLIBC_2.2) [SVID.3]
xdr_bytes(GLIBC_2.2) [SVID.3]	xdr_callhdr(GLIBC_2.2) [SVID.3]	xdr_callmsg(GLIBC_2.2) [SVID.3]	xdr_char(GLIBC_2.2) [SVID.3]
xdr_double(GLIBC_2.2) [SVID.3]	xdr_enum(GLIBC_2.2) [SVID.3]	xdr_float(GLIBC_2.2) [SVID.3]	xdr_free(GLIBC_2.2) [SVID.3]
xdr_int(GLIBC_2.2) [SVID.3]	xdr_long(GLIBC_2.2) [SVID.3]	xdr_opaque(GLIBC_2.2) [SVID.3]	xdr_opaque_auth(GLIBC_2.2) [SVID.3]
xdr_pointer(GLIBC_2.2) [SVID.3]	xdr_reference(GLIBC_2.2) [SVID.3]	xdr_rejected_reply(GLIBC_2.2) [SVID.3]	xdr_replymsg(GLIBC_2.2) [SVID.3]
xdr_short(GLIBC_2.2) [SVID.3]	xdr_string(GLIBC_2.2) [SVID.3]	xdr_u_char(GLIBC_2.2) [SVID.3]	xdr_u_int(GLIBC_2.2) [LSB]
xdr_u_long(GLIBC_2.2) [SVID.3]	xdr_u_short(GLIBC_2.2) [SVID.3]	xdr_union(GLIBC_2.2) [SVID.3]	xdr_vector(GLIBC_2.2) [SVID.3]
xdr_void(GLIBC_2.2) [SVID.3]	xdr_wrapstring(GLIBC_2.2) [SVID.3]	xdrmem_create(GLIBC_2.2) [SVID.3]	xdrrec_create(GLIBC_2.2) [SVID.3]
xdrrec_eof(GLIBC_2.2) [SVID.3]			

19

## 11.2.2 System Calls

20

### 11.2.2.1 Interfaces for System Calls

21

An LSB conforming implementation shall provide the architecture specific functions for System Calls specified in Table 11-3, with the full mandatory functionality as described in the referenced underlying specification.

22

23

24

**Table 11-3 libc - System Calls Function Interfaces**

__fxstat(GLIBC_2.2) [LSB]	__getpgid(GLIBC_2.2) [LSB]	__lxstat(GLIBC_2.2) [LSB]	__xmknod(GLIBC_2.2) [LSB]
__xstat(GLIBC_2.2) [LSB]	access(GLIBC_2.2) [SUSv3]	acct(GLIBC_2.2) [LSB]	alarm(GLIBC_2.2) [SUSv3]
brk(GLIBC_2.2) [SUSv2]	chdir(GLIBC_2.2) [SUSv3]	chmod(GLIBC_2.2) [SUSv3]	chown(GLIBC_2.2) [SUSv3]
chroot(GLIBC_2.2)	clock(GLIBC_2.2)	close(GLIBC_2.2)	closedir(GLIBC_2.2)

) [SUSv2]	[SUSv3]	[SUSv3]	2) [SUSv3]
creat(GLIBC_2.2) [SUSv3]	dup(GLIBC_2.2) [SUSv3]	dup2(GLIBC_2.2) [SUSv3]	execl(GLIBC_2.2) [SUSv3]
execle(GLIBC_2.2) [SUSv3]	execlp(GLIBC_2.2) [SUSv3]	execv(GLIBC_2.2) [SUSv3]	execve(GLIBC_2.2) [SUSv3]
execvp(GLIBC_2.2) [SUSv3]	exit(GLIBC_2.2) [SUSv3]	fchdir(GLIBC_2.2) [SUSv3]	fchmod(GLIBC_2.2) [SUSv3]
fchown(GLIBC_2.2) [SUSv3]	fcntl(GLIBC_2.2) [LSB]	fdatasync(GLIBC_2.2) [SUSv3]	flock(GLIBC_2.2) [LSB]
fork(GLIBC_2.2) [SUSv3]	fstatvfs(GLIBC_2.2) [SUSv3]	fsync(GLIBC_2.2) [SUSv3]	ftime(GLIBC_2.2) [SUSv3]
ftruncate(GLIBC_2.2) [SUSv3]	getcontext(GLIBC_2.2) [SUSv3]	getegid(GLIBC_2.2) [SUSv3]	geteuid(GLIBC_2.2) [SUSv3]
getgid(GLIBC_2.2) [SUSv3]	getgroups(GLIBC_2.2) [SUSv3]	getitimer(GLIBC_2.2) [SUSv3]	getloadavg(GLIBC_2.2) [LSB]
getpagesize(GLIBC_2.2) [SUSv2]	getpgid(GLIBC_2.2) [SUSv3]	getpgrp(GLIBC_2.2) [SUSv3]	getpid(GLIBC_2.2) [SUSv3]
getppid(GLIBC_2.2) [SUSv3]	getpriority(GLIBC_2.2) [SUSv3]	getrlimit(GLIBC_2.2) [SUSv3]	getrusage(GLIBC_2.2) [SUSv3]
getsid(GLIBC_2.2) [SUSv3]	getuid(GLIBC_2.2) [SUSv3]	getwd(GLIBC_2.2) [SUSv3]	initgroups(GLIBC_2.2) [LSB]
ioctl(GLIBC_2.2) [LSB]	kill(GLIBC_2.2) [LSB]	killpg(GLIBC_2.2) [SUSv3]	lchown(GLIBC_2.2) [SUSv3]
link(GLIBC_2.2) [LSB]	lockf(GLIBC_2.2) [SUSv3]	lseek(GLIBC_2.2) [SUSv3]	mkdir(GLIBC_2.2) [SUSv3]
mkfifo(GLIBC_2.2) [SUSv3]	mlock(GLIBC_2.2) [SUSv3]	mlockall(GLIBC_2.2) [SUSv3]	mmap(GLIBC_2.2) [SUSv3]
mprotect(GLIBC_2.2) [SUSv3]	msync(GLIBC_2.2) [SUSv3]	munlock(GLIBC_2.2) [SUSv3]	munlockall(GLIBC_2.2) [SUSv3]
munmap(GLIBC_2.2) [SUSv3]	nanosleep(GLIBC_2.2) [SUSv3]	nice(GLIBC_2.2) [SUSv3]	open(GLIBC_2.2) [SUSv3]
opendir(GLIBC_2.2) [SUSv3]	pathconf(GLIBC_2.2) [SUSv3]	pause(GLIBC_2.2) [SUSv3]	pipe(GLIBC_2.2) [SUSv3]
poll(GLIBC_2.2) [SUSv3]	read(GLIBC_2.2) [SUSv3]	readdir(GLIBC_2.2) [SUSv3]	readdir_r(GLIBC_2.2) [SUSv3]
readlink(GLIBC_2.2) [SUSv3]	readv(GLIBC_2.2) [SUSv3]	rename(GLIBC_2.2) [SUSv3]	rmdir(GLIBC_2.2) [SUSv3]
sbrk(GLIBC_2.2) [SUSv2]	sched_get_priority_max(GLIBC_2.2) [SUSv3]	sched_get_priority_min(GLIBC_2.2) [SUSv3]	sched_getparam(GLIBC_2.2) [SUSv3]
sched_getschedul	sched_rr_get_inte	sched_setparam(	sched_setschedule

er(GLIBC_2.2) [SUSv3]	rval(GLIBC_2.2) [SUSv3]	GLIBC_2.2) [SUSv3]	r(GLIBC_2.2) [SUSv3]
sched_yield(GLIBC_2.2) [SUSv3]	select(GLIBC_2.2) [SUSv3]	setcontext(GLIBC_2.2) [SUSv3]	setegid(GLIBC_2.2) [SUSv3]
seteuid(GLIBC_2.2) [SUSv3]	setgid(GLIBC_2.2) [SUSv3]	setitimer(GLIBC_2.2) [SUSv3]	setpgid(GLIBC_2.2) [SUSv3]
setpgrp(GLIBC_2.2) [SUSv3]	setpriority(GLIBC_2.2) [SUSv3]	setregid(GLIBC_2.2) [SUSv3]	setreuid(GLIBC_2.2) [SUSv3]
setrlimit(GLIBC_2.2) [SUSv3]	setrlimit64(GLIBC_2.2) [LFS]	setsid(GLIBC_2.2) [SUSv3]	setuid(GLIBC_2.2) [SUSv3]
sleep(GLIBC_2.2) [SUSv3]	statvfs(GLIBC_2.2) [SUSv3]	stime(GLIBC_2.2) [LSB]	symlink(GLIBC_2.2) [SUSv3]
sync(GLIBC_2.2) [SUSv3]	sysconf(GLIBC_2.2) [SUSv3]	time(GLIBC_2.2) [SUSv3]	times(GLIBC_2.2) [SUSv3]
truncate(GLIBC_2.2) [SUSv3]	ulimit(GLIBC_2.2) [SUSv3]	umask(GLIBC_2.2) [SUSv3]	uname(GLIBC_2.2) [SUSv3]
unlink(GLIBC_2.2) [LSB]	utime(GLIBC_2.2) [SUSv3]	utimes(GLIBC_2.2) [SUSv3]	vfork(GLIBC_2.2) [SUSv3]
wait(GLIBC_2.2) [SUSv3]	wait4(GLIBC_2.2) [LSB]	waitpid(GLIBC_2.2) [LSB]	write(GLIBC_2.2) [SUSv3]
writev(GLIBC_2.2) [SUSv3]			

25

### 11.2.3 Standard I/O

26

#### 11.2.3.1 Interfaces for Standard I/O

27

An LSB conforming implementation shall provide the architecture specific functions for Standard I/O specified in Table 11-4, with the full mandatory functionality as described in the referenced underlying specification.

28

29

30

**Table 11-4 libc - Standard I/O Function Interfaces**

_IO_feof(GLIBC_2.2) [LSB]	_IO_getc(GLIBC_2.2) [LSB]	_IO_putc(GLIBC_2.2) [LSB]	_IO_puts(GLIBC_2.2) [LSB]
asprintf(GLIBC_2.2) [LSB]	clearerr(GLIBC_2.2) [SUSv3]	ctermid(GLIBC_2.2) [SUSv3]	fclose(GLIBC_2.2) [SUSv3]
fdopen(GLIBC_2.2) [SUSv3]	feof(GLIBC_2.2) [SUSv3]	ferror(GLIBC_2.2) [SUSv3]	fflush(GLIBC_2.2) [SUSv3]
fflush_unlocked(GLIBC_2.2) [LSB]	fgetc(GLIBC_2.2) [SUSv3]	fgetpos(GLIBC_2.2) [SUSv3]	fgets(GLIBC_2.2) [SUSv3]
fgetwc_unlocked(GLIBC_2.2) [LSB]	fileno(GLIBC_2.2) [SUSv3]	flockfile(GLIBC_2.2) [SUSv3]	fopen(GLIBC_2.2) [SUSv3]
fprintf(GLIBC_2.2) [SUSv3]	fputc(GLIBC_2.2) [SUSv3]	fputs(GLIBC_2.2) [SUSv3]	fread(GLIBC_2.2) [SUSv3]

freopen(GLIBC_2.2) [SUSv3]	fscanf(GLIBC_2.2) [LSB]	fseek(GLIBC_2.2) [SUSv3]	fseeko(GLIBC_2.2) [SUSv3]
fsetpos(GLIBC_2.2) [SUSv3]	ftell(GLIBC_2.2) [SUSv3]	ftello(GLIBC_2.2) [SUSv3]	fwrite(GLIBC_2.2) [SUSv3]
getc(GLIBC_2.2) [SUSv3]	getc_unlocked(GLIBC_2.2) [SUSv3]	getchar(GLIBC_2.2) [SUSv3]	getchar_unlocked(GLIBC_2.2) [SUSv3]
getw(GLIBC_2.2) [SUSv2]	pclose(GLIBC_2.2) [SUSv3]	popen(GLIBC_2.2) [SUSv3]	printf(GLIBC_2.2) [SUSv3]
putc(GLIBC_2.2) [SUSv3]	putc_unlocked(GLIBC_2.2) [SUSv3]	putchar(GLIBC_2.2) [SUSv3]	putchar_unlocked(GLIBC_2.2) [SUSv3]
puts(GLIBC_2.2) [SUSv3]	putw(GLIBC_2.2) [SUSv2]	remove(GLIBC_2.2) [SUSv3]	rewind(GLIBC_2.2) [SUSv3]
rewinddir(GLIBC_2.2) [SUSv3]	scanf(GLIBC_2.2) [LSB]	seekdir(GLIBC_2.2) [SUSv3]	setbuf(GLIBC_2.2) [SUSv3]
setbuffer(GLIBC_2.2) [LSB]	setvbuf(GLIBC_2.2) [SUSv3]	snprintf(GLIBC_2.2) [SUSv3]	sprintf(GLIBC_2.2) [SUSv3]
sscanf(GLIBC_2.2) [LSB]	telldir(GLIBC_2.2) [SUSv3]	tempnam(GLIBC_2.2) [SUSv3]	ungetc(GLIBC_2.2) [SUSv3]
vasprintf(GLIBC_2.2) [LSB]	vdprintf(GLIBC_2.2) [LSB]	vfprintf(GLIBC_2.2) [SUSv3]	vprintf(GLIBC_2.2) [SUSv3]
vsnprintf(GLIBC_2.2) [SUSv3]	vsprintf(GLIBC_2.2) [SUSv3]		

31

32

33

34

An LSB conforming implementation shall provide the architecture specific data interfaces for Standard I/O specified in Table 11-5, with the full mandatory functionality as described in the referenced underlying specification.

35

**Table 11-5 libc - Standard I/O Data Interfaces**

stderr(GLIBC_2.2) [SUSv3]	stdin(GLIBC_2.2) [SUSv3]	stdout(GLIBC_2.2) [SUSv3]	
---------------------------	--------------------------	---------------------------	--

36

## 11.2.4 Signal Handling

37

### 11.2.4.1 Interfaces for Signal Handling

38

39

40

An LSB conforming implementation shall provide the architecture specific functions for Signal Handling specified in Table 11-6, with the full mandatory functionality as described in the referenced underlying specification.

41

**Table 11-6 libc - Signal Handling Function Interfaces**

__libc_current_sigrtmax(GLIBC_2.2) [LSB]	__libc_current_sigrtmin(GLIBC_2.2) [LSB]	__sigsetjmp(GLIBC_2.2) [LSB]	__sysv_signal(GLIBC_2.2) [LSB]
--	--	------------------------------	--------------------------------

bsd_signal(GLIBC_2.2) [SUSv3]	psignal(GLIBC_2.2) [LSB]	raise(GLIBC_2.2) [SUSv3]	sigaction(GLIBC_2.2) [SUSv3]
sigaddset(GLIBC_2.2) [SUSv3]	sigaltstack(GLIBC_2.2) [SUSv3]	sigandset(GLIBC_2.2) [LSB]	sigdelset(GLIBC_2.2) [SUSv3]
sigemptyset(GLIBC_2.2) [SUSv3]	sigfillset(GLIBC_2.2) [SUSv3]	sighold(GLIBC_2.2) [SUSv3]	sigignore(GLIBC_2.2) [SUSv3]
siginterrupt(GLIBC_2.2) [SUSv3]	sigisemptyset(GLIBC_2.2) [LSB]	sigismember(GLIBC_2.2) [SUSv3]	siglongjmp(GLIBC_2.2) [SUSv3]
signal(GLIBC_2.2) [SUSv3]	sigorset(GLIBC_2.2) [LSB]	sigpause(GLIBC_2.2) [SUSv3]	sigpending(GLIBC_2.2) [SUSv3]
sigprocmask(GLIBC_2.2) [SUSv3]	sigqueue(GLIBC_2.2) [SUSv3]	sigrelse(GLIBC_2.2) [SUSv3]	sigreturn(GLIBC_2.2) [LSB]
sigset(GLIBC_2.2) [SUSv3]	sigsuspend(GLIBC_2.2) [SUSv3]	sigtimedwait(GLIBC_2.2) [SUSv3]	sigwait(GLIBC_2.2) [SUSv3]
sigwaitinfo(GLIBC_2.2) [SUSv3]			

42

An LSB conforming implementation shall provide the architecture specific data interfaces for Signal Handling specified in Table 11-7, with the full mandatory functionality as described in the referenced underlying specification.

43

44

45

**Table 11-7 libc - Signal Handling Data Interfaces**

46

_sys_siglist(GLIBC_2.3.3) [LSB]			
---------------------------------	--	--	--

47

### 11.2.5 Localization Functions

48

#### 11.2.5.1 Interfaces for Localization Functions

49

An LSB conforming implementation shall provide the architecture specific functions for Localization Functions specified in Table 11-8, with the full mandatory functionality as described in the referenced underlying specification.

50

51

**Table 11-8 libc - Localization Functions Function Interfaces**

52

bind_textdomain_codeset(GLIBC_2.2) [LSB]	bindtextdomain(GLIBC_2.2) [LSB]	catclose(GLIBC_2.2) [SUSv3]	catgets(GLIBC_2.2) [SUSv3]
catopen(GLIBC_2.2) [SUSv3]	dcgettext(GLIBC_2.2) [LSB]	dcngettext(GLIBC_2.2) [LSB]	dgettext(GLIBC_2.2) [LSB]
dcngettext(GLIBC_2.2) [LSB]	gettext(GLIBC_2.2) [LSB]	iconv(GLIBC_2.2) [SUSv3]	iconv_close(GLIBC_2.2) [SUSv3]
iconv_open(GLIBC_2.2) [SUSv3]	localeconv(GLIBC_2.2) [SUSv3]	ngettext(GLIBC_2.2) [LSB]	nl_langinfo(GLIBC_2.2) [SUSv3]
setlocale(GLIBC_2.2) [SUSv3]	textdomain(GLIBC_2.2) [LSB]		

53

54 An LSB conforming implementation shall provide the architecture specific data  
 55 interfaces for Localization Functions specified in Table 11-9, with the full mandatory  
 56 functionality as described in the referenced underlying specification.

57 **Table 11-9 libc - Localization Functions Data Interfaces**

_nl_msg_cat_cntr( GLIBC_2.2) [LSB]			
---------------------------------------	--	--	--

58

## 11.2.6 Socket Interface

### 11.2.6.1 Interfaces for Socket Interface

59 An LSB conforming implementation shall provide the architecture specific functions  
 60 for Socket Interface specified in Table 11-10, with the full mandatory functionality as  
 61 described in the referenced underlying specification.  
 62

63 **Table 11-10 libc - Socket Interface Function Interfaces**

__h_errno_locatio n(GLIBC_2.2) [LSB]	accept(GLIBC_2.2 ) [SUSv3]	bind(GLIBC_2.2) [SUSv3]	bindresvport(GLI BC_2.2) [LSB]
connect(GLIBC_2. 2) [SUSv3]	gethostid(GLIBC_ 2.2) [SUSv3]	gethostname(GLI BC_2.2) [SUSv3]	getpeername(GLI BC_2.2) [SUSv3]
getsockname(GLI BC_2.2) [SUSv3]	getsockopt(GLIBC _2.2) [LSB]	if_freenameindex( GLIBC_2.2) [SUSv3]	if_indextoname(G LIBC_2.2) [SUSv3]
if_nameindex(GLI BC_2.2) [SUSv3]	if_nametoindex(G LIBC_2.2) [SUSv3]	listen(GLIBC_2.2) [SUSv3]	recv(GLIBC_2.2) [SUSv3]
recvfrom(GLIBC_ 2.2) [SUSv3]	recvmsg(GLIBC_2 .2) [SUSv3]	send(GLIBC_2.2) [SUSv3]	sendmsg(GLIBC_ 2.2) [SUSv3]
sendto(GLIBC_2.2 ) [SUSv3]	setsockopt(GLIBC _2.2) [LSB]	shutdown(GLIBC _2.2) [SUSv3]	socketatmark(GLIB C_2.2.4) [SUSv3]
socket(GLIBC_2.2 ) [SUSv3]	socketpair(GLIBC _2.2) [SUSv3]		

64

## 11.2.7 Wide Characters

### 11.2.7.1 Interfaces for Wide Characters

65 An LSB conforming implementation shall provide the architecture specific functions  
 66 for Wide Characters specified in Table 11-11, with the full mandatory functionality  
 67 as described in the referenced underlying specification.  
 68

69 **Table 11-11 libc - Wide Characters Function Interfaces**

__wcstod_internal (GLIBC_2.2) [LSB]	__wcstof_internal( GLIBC_2.2) [LSB]	__wcstol_internal( GLIBC_2.2) [LSB]	__wcstold_interna l(GLIBC_2.2) [LSB]
__wcstoul_interna l(GLIBC_2.2)	btowc(GLIBC_2.2) [SUSv3]	fgetwc(GLIBC_2.2 ) [SUSv3]	fgetws(GLIBC_2.2 ) [SUSv3]

[LSB]			
fputwc(GLIBC_2.2) [SUSv3]	fputws(GLIBC_2.2) [SUSv3]	fwide(GLIBC_2.2) [SUSv3]	fwprintf(GLIBC_2.2) [SUSv3]
fwscanf(GLIBC_2.2) [LSB]	getwc(GLIBC_2.2) [SUSv3]	getwchar(GLIBC_2.2) [SUSv3]	mblen(GLIBC_2.2) [SUSv3]
mbrlen(GLIBC_2.2) [SUSv3]	mbrtowc(GLIBC_2.2) [SUSv3]	mbsinit(GLIBC_2.2) [SUSv3]	mbsnrtowcs(GLIBC_2.2) [LSB]
mbsrtowcs(GLIBC_2.2) [SUSv3]	mbstowcs(GLIBC_2.2) [SUSv3]	mbtowc(GLIBC_2.2) [SUSv3]	putwc(GLIBC_2.2) [SUSv3]
putwchar(GLIBC_2.2) [SUSv3]	swprintf(GLIBC_2.2) [SUSv3]	swscanf(GLIBC_2.2) [LSB]	towctrans(GLIBC_2.2) [SUSv3]
tolower(GLIBC_2.2) [SUSv3]	toupper(GLIBC_2.2) [SUSv3]	ungetwc(GLIBC_2.2) [SUSv3]	vfwprintf(GLIBC_2.2) [SUSv3]
vfwscanf(GLIBC_2.2) [LSB]	vswprintf(GLIBC_2.2) [SUSv3]	vswscanf(GLIBC_2.2) [LSB]	vwprintf(GLIBC_2.2) [SUSv3]
vwscanf(GLIBC_2.2) [LSB]	wcpcpy(GLIBC_2.2) [LSB]	wcpncpy(GLIBC_2.2) [LSB]	wcrtomb(GLIBC_2.2) [SUSv3]
wscasecmp(GLIBC_2.2) [LSB]	wscat(GLIBC_2.2) [SUSv3]	wcschr(GLIBC_2.2) [SUSv3]	wscmp(GLIBC_2.2) [SUSv3]
wscoll(GLIBC_2.2) [SUSv3]	wcscpy(GLIBC_2.2) [SUSv3]	wcscspn(GLIBC_2.2) [SUSv3]	wcsdup(GLIBC_2.2) [LSB]
wcsftime(GLIBC_2.2) [SUSv3]	wcslen(GLIBC_2.2) [SUSv3]	wcsncasecmp(GLIBC_2.2) [LSB]	wcsncat(GLIBC_2.2) [SUSv3]
wcsncmp(GLIBC_2.2) [SUSv3]	wcsncpy(GLIBC_2.2) [SUSv3]	wcsnlen(GLIBC_2.2) [LSB]	wcsnrtombs(GLIBC_2.2) [LSB]
wcspbrk(GLIBC_2.2) [SUSv3]	wcsrchr(GLIBC_2.2) [SUSv3]	wcsrtombs(GLIBC_2.2) [SUSv3]	wcsspn(GLIBC_2.2) [SUSv3]
wcsstr(GLIBC_2.2) [SUSv3]	wcstod(GLIBC_2.2) [SUSv3]	wcstof(GLIBC_2.2) [SUSv3]	wcstoimax(GLIBC_2.2) [SUSv3]
wcstok(GLIBC_2.2) [SUSv3]	wcstol(GLIBC_2.2) [SUSv3]	wcstold(GLIBC_2.2) [SUSv3]	wcstoll(GLIBC_2.2) [SUSv3]
wcstombs(GLIBC_2.2) [SUSv3]	wcstoq(GLIBC_2.2) [LSB]	wcstoul(GLIBC_2.2) [SUSv3]	wcstoull(GLIBC_2.2) [SUSv3]
wcstoumax(GLIBC_2.2) [SUSv3]	wcstouq(GLIBC_2.2) [LSB]	wcswcs(GLIBC_2.2) [SUSv3]	wcswidth(GLIBC_2.2) [SUSv3]
wcsxfrm(GLIBC_2.2) [SUSv3]	wctob(GLIBC_2.2) [SUSv3]	wctomb(GLIBC_2.2) [SUSv3]	wctrans(GLIBC_2.2) [SUSv3]
wctype(GLIBC_2.2) [SUSv3]	wcwidth(GLIBC_2.2) [SUSv3]	wmemchr(GLIBC_2.2) [SUSv3]	wmemcmp(GLIBC_2.2) [SUSv3]
wmemcpy(GLIBC_2.2) [SUSv3]	wmemmove(GLIBC_2.2) [SUSv3]	wmemset(GLIBC_2.2) [SUSv3]	wprintf(GLIBC_2.2) [SUSv3]



70

wscanf(GLIBC_2.2) [LSB]			
-------------------------	--	--	--

## 11.2.8 String Functions

71

### 11.2.8.1 Interfaces for String Functions

72

An LSB conforming implementation shall provide the architecture specific functions for String Functions specified in Table 11-12, with the full mandatory functionality as described in the referenced underlying specification.

73

74

75

**Table 11-12 libc - String Functions Function Interfaces**

__mempcpy(GLIBC_2.2) [LSB]	__rawmemchr(GLIBC_2.2) [LSB]	__stpcpy(GLIBC_2.2) [LSB]	__strdup(GLIBC_2.2) [LSB]
__strtod_internal(GLIBC_2.2) [LSB]	__strtof_internal(GLIBC_2.2) [LSB]	__strtok_r(GLIBC_2.2) [LSB]	__strtol_internal(GLIBC_2.2) [LSB]
__strtold_internal(GLIBC_2.2) [LSB]	__strtoll_internal(GLIBC_2.2) [LSB]	__strtoul_internal(GLIBC_2.2) [LSB]	__strtoull_internal(GLIBC_2.2) [LSB]
bcmp(GLIBC_2.2) [SUSv3]	bcopy(GLIBC_2.2) [SUSv3]	bzero(GLIBC_2.2) [SUSv3]	ffs(GLIBC_2.2) [SUSv3]
index(GLIBC_2.2) [SUSv3]	memccpy(GLIBC_2.2) [SUSv3]	memchr(GLIBC_2.2) [SUSv3]	memcmp(GLIBC_2.2) [SUSv3]
memcpy(GLIBC_2.2) [SUSv3]	memmove(GLIBC_2.2) [SUSv3]	memrchr(GLIBC_2.2) [LSB]	memset(GLIBC_2.2) [SUSv3]
rindex(GLIBC_2.2) [SUSv3]	stpcpy(GLIBC_2.2) [LSB]	stpncpy(GLIBC_2.2) [LSB]	strcasemp(GLIBC_2.2) [SUSv3]
strcasestr(GLIBC_2.2) [LSB]	strcat(GLIBC_2.2) [SUSv3]	strchr(GLIBC_2.2) [SUSv3]	strcmp(GLIBC_2.2) [SUSv3]
strcoll(GLIBC_2.2) [SUSv3]	strcpy(GLIBC_2.2) [SUSv3]	strcspn(GLIBC_2.2) [SUSv3]	strdup(GLIBC_2.2) [SUSv3]
strerror(GLIBC_2.2) [SUSv3]	strerror_r(GLIBC_2.2) [LSB]	strfmon(GLIBC_2.2) [SUSv3]	strftime(GLIBC_2.2) [SUSv3]
strlen(GLIBC_2.2) [SUSv3]	strncasemp(GLIBC_2.2) [SUSv3]	strncat(GLIBC_2.2) [SUSv3]	strncmp(GLIBC_2.2) [SUSv3]
strncpy(GLIBC_2.2) [SUSv3]	strndup(GLIBC_2.2) [LSB]	strnlen(GLIBC_2.2) [LSB]	strpbrk(GLIBC_2.2) [SUSv3]
strptime(GLIBC_2.2) [LSB]	strrchr(GLIBC_2.2) [SUSv3]	strsep(GLIBC_2.2) [LSB]	strsignal(GLIBC_2.2) [LSB]
strspn(GLIBC_2.2) [SUSv3]	strstr(GLIBC_2.2) [SUSv3]	strtof(GLIBC_2.2) [SUSv3]	strtoimax(GLIBC_2.2) [SUSv3]
strtok(GLIBC_2.2) [SUSv3]	strtok_r(GLIBC_2.2) [SUSv3]	strtold(GLIBC_2.2) [SUSv3]	strtoll(GLIBC_2.2) [SUSv3]
strtoq(GLIBC_2.2) [LSB]	strtoull(GLIBC_2.2) [SUSv3]	strtoumax(GLIBC_2.2) [SUSv3]	strtouq(GLIBC_2.2) [LSB]

76

strxfrm(GLIBC_2.2) [SUSv3]	swab(GLIBC_2.2) [SUSv3]		
----------------------------	-------------------------	--	--

## 11.2.9 IPC Functions

77

### 11.2.9.1 Interfaces for IPC Functions

78

An LSB conforming implementation shall provide the architecture specific functions for IPC Functions specified in Table 11-13, with the full mandatory functionality as described in the referenced underlying specification.

79

80

81

**Table 11-13 libc - IPC Functions Function Interfaces**

ftok(GLIBC_2.2) [SUSv3]	msgctl(GLIBC_2.2) [SUSv3]	msgget(GLIBC_2.2) [SUSv3]	msgrcv(GLIBC_2.2) [SUSv3]
msgsnd(GLIBC_2.2) [SUSv3]	semctl(GLIBC_2.2) [SUSv3]	semget(GLIBC_2.2) [SUSv3]	semop(GLIBC_2.2) [SUSv3]
shmat(GLIBC_2.2) [SUSv3]	shmctl(GLIBC_2.2) [SUSv3]	shmdt(GLIBC_2.2) [SUSv3]	shmget(GLIBC_2.2) [SUSv3]

82

## 11.2.10 Regular Expressions

83

### 11.2.10.1 Interfaces for Regular Expressions

84

An LSB conforming implementation shall provide the architecture specific functions for Regular Expressions specified in Table 11-14, with the full mandatory functionality as described in the referenced underlying specification.

85

86

87

**Table 11-14 libc - Regular Expressions Function Interfaces**

regcomp(GLIBC_2.2) [SUSv3]	regerror(GLIBC_2.2) [SUSv3]	regexexec(GLIBC_2.3.4) [LSB]	regfree(GLIBC_2.2) [SUSv3]
----------------------------	-----------------------------	------------------------------	----------------------------

88

## 11.2.11 Character Type Functions

89

### 11.2.11.1 Interfaces for Character Type Functions

90

An LSB conforming implementation shall provide the architecture specific functions for Character Type Functions specified in Table 11-15, with the full mandatory functionality as described in the referenced underlying specification.

91

92

93

**Table 11-15 libc - Character Type Functions Function Interfaces**

__ctype_get_mb_cur_max(GLIBC_2.2) [LSB]	_tolower(GLIBC_2.2) [SUSv3]	_toupper(GLIBC_2.2) [SUSv3]	isalnum(GLIBC_2.2) [SUSv3]
isalpha(GLIBC_2.2) [SUSv3]	isascii(GLIBC_2.2) [SUSv3]	iscntrl(GLIBC_2.2) [SUSv3]	isdigit(GLIBC_2.2) [SUSv3]
isgraph(GLIBC_2.2) [SUSv3]	islower(GLIBC_2.2) [SUSv3]	isprint(GLIBC_2.2) [SUSv3]	ispunct(GLIBC_2.2) [SUSv3]
isspace(GLIBC_2.2) [SUSv3]	isupper(GLIBC_2.2) [SUSv3]	iswalnum(GLIBC_2.2) [SUSv3]	iswalpha(GLIBC_2.2) [SUSv3]

iswblank(GLIBC_2.2) [SUSv3]	iswcntrl(GLIBC_2.2) [SUSv3]	iswctype(GLIBC_2.2) [SUSv3]	iswdigit(GLIBC_2.2) [SUSv3]
iswgraph(GLIBC_2.2) [SUSv3]	iswlower(GLIBC_2.2) [SUSv3]	iswprint(GLIBC_2.2) [SUSv3]	iswpunct(GLIBC_2.2) [SUSv3]
iswspace(GLIBC_2.2) [SUSv3]	iswupper(GLIBC_2.2) [SUSv3]	iswxdigit(GLIBC_2.2) [SUSv3]	isxdigit(GLIBC_2.2) [SUSv3]
toascii(GLIBC_2.2) [SUSv3]	tolower(GLIBC_2.2) [SUSv3]	toupper(GLIBC_2.2) [SUSv3]	

94

## 11.2.12 Time Manipulation

95

### 11.2.12.1 Interfaces for Time Manipulation

96

An LSB conforming implementation shall provide the architecture specific functions for Time Manipulation specified in Table 11-16, with the full mandatory functionality as described in the referenced underlying specification.

97

98

99

**Table 11-16 libc - Time Manipulation Function Interfaces**

adjtime(GLIBC_2.2) [LSB]	asctime(GLIBC_2.2) [SUSv3]	asctime_r(GLIBC_2.2) [SUSv3]	ctime(GLIBC_2.2) [SUSv3]
ctime_r(GLIBC_2.2) [SUSv3]	difftime(GLIBC_2.2) [SUSv3]	gmtime(GLIBC_2.2) [SUSv3]	gmtime_r(GLIBC_2.2) [SUSv3]
localtime(GLIBC_2.2) [SUSv3]	localtime_r(GLIBC_2.2) [SUSv3]	mktime(GLIBC_2.2) [SUSv3]	tzset(GLIBC_2.2) [SUSv3]
ualarm(GLIBC_2.2) [SUSv3]			

100

101

102

103

An LSB conforming implementation shall provide the architecture specific data interfaces for Time Manipulation specified in Table 11-17, with the full mandatory functionality as described in the referenced underlying specification.

104

**Table 11-17 libc - Time Manipulation Data Interfaces**

__daylight(GLIBC_2.2) [LSB]	__timezone(GLIBC_2.2) [LSB]	__tzname(GLIBC_2.2) [LSB]	daylight(GLIBC_2.2) [SUSv3]
timezone(GLIBC_2.2) [SUSv3]	tzname(GLIBC_2.2) [SUSv3]		

105

## 11.2.13 Terminal Interface Functions

106

### 11.2.13.1 Interfaces for Terminal Interface Functions

107

108

109

An LSB conforming implementation shall provide the architecture specific functions for Terminal Interface Functions specified in Table 11-18, with the full mandatory functionality as described in the referenced underlying specification.

110

**Table 11-18 libc - Terminal Interface Functions Function Interfaces**

cfgetispeed(GLIB)	cfgetospeed(GLIB)	cfmakeraw(GLIB)	cfsetispeed(GLIB)
-------------------	-------------------	-----------------	-------------------

111

C_2.2) [SUSv3]	C_2.2) [SUSv3]	C_2.2) [LSB]	C_2.2) [SUSv3]
cfsetospeed(GLIBC_C_2.2) [SUSv3]	cfsetospeed(GLIBC_2.2) [LSB]	tcdrain(GLIBC_2.2) [SUSv3]	tcflow(GLIBC_2.2) [SUSv3]
tcflush(GLIBC_2.2) [SUSv3]	tcgetattr(GLIBC_2.2) [SUSv3]	tcgetpgrp(GLIBC_2.2) [SUSv3]	tcgetsid(GLIBC_2.2) [SUSv3]
tcsendbreak(GLIBC_C_2.2) [SUSv3]	tcsetattr(GLIBC_2.2) [SUSv3]	tcsetpgrp(GLIBC_2.2) [SUSv3]	

## 11.2.14 System Database Interface

112

### 11.2.14.1 Interfaces for System Database Interface

113

An LSB conforming implementation shall provide the architecture specific functions for System Database Interface specified in Table 11-19, with the full mandatory functionality as described in the referenced underlying specification.

114

115

116

**Table 11-19 libc - System Database Interface Function Interfaces**

endgrent(GLIBC_2.2) [SUSv3]	endprotoent(GLIBC_C_2.2) [SUSv3]	endpwent(GLIBC_2.2) [SUSv3]	endservent(GLIBC_C_2.2) [SUSv3]
endutent(GLIBC_2.2) [SUSv2]	endutxent(GLIBC_2.2) [SUSv3]	getgrent(GLIBC_2.2) [SUSv3]	getgrgid(GLIBC_2.2) [SUSv3]
getgrgid_r(GLIBC_2.2) [SUSv3]	getgrnam(GLIBC_2.2) [SUSv3]	getgrnam_r(GLIBC_C_2.2) [SUSv3]	getgrouplist(GLIBC_C_2.2.4) [LSB]
gethostbyaddr(GLIBC_2.2) [SUSv3]	gethostbyname(GLIBC_2.2) [SUSv3]	getprotobyname(GLIBC_2.2) [SUSv3]	getprotobynumber(GLIBC_2.2) [SUSv3]
getprotoent(GLIBC_C_2.2) [SUSv3]	getpwent(GLIBC_2.2) [SUSv3]	getpwnam(GLIBC_2.2) [SUSv3]	getpwnam_r(GLIBC_2.2) [SUSv3]
getpwuid(GLIBC_2.2) [SUSv3]	getpwuid_r(GLIBC_C_2.2) [SUSv3]	getservbyname(GLIBC_2.2) [SUSv3]	getservbyport(GLIBC_2.2) [SUSv3]
getservent(GLIBC_2.2) [SUSv3]	getutent(GLIBC_2.2) [LSB]	getutent_r(GLIBC_2.2) [LSB]	getutxent(GLIBC_2.2) [SUSv3]
getutxid(GLIBC_2.2) [SUSv3]	getutxline(GLIBC_2.2) [SUSv3]	pututxline(GLIBC_2.2) [SUSv3]	setgrent(GLIBC_2.2) [SUSv3]
setgroups(GLIBC_2.2) [LSB]	setprotoent(GLIBC_C_2.2) [SUSv3]	setpwent(GLIBC_2.2) [SUSv3]	setservent(GLIBC_2.2) [SUSv3]
setutent(GLIBC_2.2) [LSB]	setutxent(GLIBC_2.2) [SUSv3]	utmpname(GLIBC_C_2.2) [LSB]	

117

## 11.2.15 Language Support

118

### 11.2.15.1 Interfaces for Language Support

119

An LSB conforming implementation shall provide the architecture specific functions for Language Support specified in Table 11-20, with the full mandatory functionality as described in the referenced underlying specification.

120

121

122

**Table 11-20 libc - Language Support Function Interfaces**

123

__libc_start_main(GLIBC_2.2) [LSB]			
------------------------------------	--	--	--

## 11.2.16 Large File Support

124

### 11.2.16.1 Interfaces for Large File Support

125

An LSB conforming implementation shall provide the architecture specific functions for Large File Support specified in Table 11-21, with the full mandatory functionality as described in the referenced underlying specification.

126

127

**Table 11-21 libc - Large File Support Function Interfaces**

128

__fxstat64(GLIBC_2.2) [LSB]	__lxstat64(GLIBC_2.2) [LSB]	__xstat64(GLIBC_2.2) [LSB]	creat64(GLIBC_2.2) [LFS]
fgetpos64(GLIBC_2.2) [LFS]	fopen64(GLIBC_2.2) [LFS]	freopen64(GLIBC_2.2) [LFS]	fseeko64(GLIBC_2.2) [LFS]
fsetpos64(GLIBC_2.2) [LFS]	fstatvfs64(GLIBC_2.2) [LFS]	ftello64(GLIBC_2.2) [LFS]	ftruncate64(GLIBC_2.2) [LFS]
ftw64(GLIBC_2.2) [LFS]	getrlimit64(GLIBC_2.2) [LFS]	lockf64(GLIBC_2.2) [LFS]	mkstemp64(GLIBC_2.2) [LFS]
mmap64(GLIBC_2.2) [LFS]	nftw64(GLIBC_2.3) [LFS]	readdir64(GLIBC_2.2) [LFS]	statvfs64(GLIBC_2.2) [LFS]
tmpfile64(GLIBC_2.2) [LFS]	truncate64(GLIBC_2.2) [LFS]		

129

## 11.2.17 Standard Library

130

### 11.2.17.1 Interfaces for Standard Library

131

An LSB conforming implementation shall provide the architecture specific functions for Standard Library specified in Table 11-22, with the full mandatory functionality as described in the referenced underlying specification.

132

133

134

**Table 11-22 libc - Standard Library Function Interfaces**

_Exit(GLIBC_2.2) [SUSv3]	__assert_fail(GLIBC_2.2) [LSB]	__cxa_atexit(GLIBC_2.2) [LSB]	__errno_location(GLIBC_2.2) [LSB]
__fpending(GLIBC_2.2) [LSB]	__getpagesize(GLIBC_2.2) [LSB]	__isinf(GLIBC_2.2) [LSB]	__isinff(GLIBC_2.2) [LSB]
__isinf(GLIBC_2.2) [LSB]	__isnan(GLIBC_2.2) [LSB]	__isnanf(GLIBC_2.2) [LSB]	__isnanl(GLIBC_2.2) [LSB]
__sysconf(GLIBC_2.2) [LSB]	_exit(GLIBC_2.2) [SUSv3]	_longjmp(GLIBC_2.2) [SUSv3]	_setjmp(GLIBC_2.2) [SUSv3]
a64l(GLIBC_2.2) [SUSv3]	abort(GLIBC_2.2) [SUSv3]	abs(GLIBC_2.2) [SUSv3]	atof(GLIBC_2.2) [SUSv3]
atoi(GLIBC_2.2)	atol(GLIBC_2.2)	atoll(GLIBC_2.2)	basename(GLIBC_2.2)

[SUSv3]	[SUSv3]	[SUSv3]	_2.2) [SUSv3]
bsearch(GLIBC_2.2) [SUSv3]	calloc(GLIBC_2.2) [SUSv3]	closelog(GLIBC_2.2) [SUSv3]	confstr(GLIBC_2.2) [SUSv3]
cuserid(GLIBC_2.2) [SUSv2]	daemon(GLIBC_2.2) [LSB]	dirname(GLIBC_2.2) [SUSv3]	div(GLIBC_2.2) [SUSv3]
drand48(GLIBC_2.2) [SUSv3]	ecvt(GLIBC_2.2) [SUSv3]	erand48(GLIBC_2.2) [SUSv3]	err(GLIBC_2.2) [LSB]
error(GLIBC_2.2) [LSB]	errx(GLIBC_2.2) [LSB]	fcvt(GLIBC_2.2) [SUSv3]	fmtmsg(GLIBC_2.2) [SUSv3]
fnmatch(GLIBC_2.2.3) [SUSv3]	fpathconf(GLIBC_2.2) [SUSv3]	free(GLIBC_2.2) [SUSv3]	freeaddrinfo(GLIBC_2.2) [SUSv3]
ftw(GLIBC_2.2) [SUSv3]	ftw(GLIBC_2.2) [SUSv3]	funlockfile(GLIBC_2.2) [SUSv3]	gai_strerror(GLIBC_2.2) [SUSv3]
gcvt(GLIBC_2.2) [SUSv3]	getaddrinfo(GLIBC_2.2) [SUSv3]	getcwd(GLIBC_2.2) [SUSv3]	getdate(GLIBC_2.2) [SUSv3]
getenv(GLIBC_2.2) [SUSv3]	getlogin(GLIBC_2.2) [SUSv3]	getlogin_r(GLIBC_2.2) [SUSv3]	getnameinfo(GLIBC_2.2) [SUSv3]
getopt(GLIBC_2.2) [LSB]	getopt_long(GLIBC_2.2) [LSB]	getopt_long_only(GLIBC_2.2) [LSB]	getsubopt(GLIBC_2.2) [SUSv3]
gettimeofday(GLIBC_2.2) [SUSv3]	glob(GLIBC_2.2) [SUSv3]	glob64(GLIBC_2.2) [LSB]	globfree(GLIBC_2.2) [SUSv3]
globfree64(GLIBC_2.2) [LSB]	grantpt(GLIBC_2.2) [SUSv3]	hcreate(GLIBC_2.2) [SUSv3]	hdestroy(GLIBC_2.2) [SUSv3]
hsearch(GLIBC_2.2) [SUSv3]	htonl(GLIBC_2.2) [SUSv3]	htons(GLIBC_2.2) [SUSv3]	imaxabs(GLIBC_2.2) [SUSv3]
imaxdiv(GLIBC_2.2) [SUSv3]	inet_addr(GLIBC_2.2) [SUSv3]	inet_ntoa(GLIBC_2.2) [SUSv3]	inet_ntop(GLIBC_2.2) [SUSv3]
inet_pton(GLIBC_2.2) [SUSv3]	initstate(GLIBC_2.2) [SUSv3]	insque(GLIBC_2.2) [SUSv3]	isatty(GLIBC_2.2) [SUSv3]
isblank(GLIBC_2.2) [SUSv3]	jrand48(GLIBC_2.2) [SUSv3]	l64a(GLIBC_2.2) [SUSv3]	labs(GLIBC_2.2) [SUSv3]
lcong48(GLIBC_2.2) [SUSv3]	ldiv(GLIBC_2.2) [SUSv3]	lfind(GLIBC_2.2) [SUSv3]	llabs(GLIBC_2.2) [SUSv3]
lldiv(GLIBC_2.2) [SUSv3]	longjmp(GLIBC_2.2) [SUSv3]	lrand48(GLIBC_2.2) [SUSv3]	lsearch(GLIBC_2.2) [SUSv3]
makecontext(GLIBC_2.2) [SUSv3]	malloc(GLIBC_2.2) [SUSv3]	memmem(GLIBC_2.2) [LSB]	mkstemp(GLIBC_2.2) [SUSv3]
mktemp(GLIBC_2.2) [SUSv3]	mrnd48(GLIBC_2.2) [SUSv3]	nftw(GLIBC_2.3.3) [SUSv3]	nrnd48(GLIBC_2.2) [SUSv3]
ntohl(GLIBC_2.2) [SUSv3]	ntohs(GLIBC_2.2) [SUSv3]	openlog(GLIBC_2.2) [SUSv3]	perror(GLIBC_2.2) [SUSv3]

posix_memalign(GLIBC_2.2) [SUSv3]	posix_openpt(GLIBC_2.2.1) [SUSv3]	ptsname(GLIBC_2.2) [SUSv3]	putenv(GLIBC_2.2) [SUSv3]
qsort(GLIBC_2.2) [SUSv3]	rand(GLIBC_2.2) [SUSv3]	rand_r(GLIBC_2.2) [SUSv3]	random(GLIBC_2.2) [SUSv3]
realloc(GLIBC_2.2) [SUSv3]	realpath(GLIBC_2.3) [SUSv3]	remque(GLIBC_2.2) [SUSv3]	seed48(GLIBC_2.2) [SUSv3]
setenv(GLIBC_2.2) [SUSv3]	sethostname(GLIBC_2.2) [LSB]	setlogmask(GLIBC_2.2) [SUSv3]	setstate(GLIBC_2.2) [SUSv3]
srand(GLIBC_2.2) [SUSv3]	srand48(GLIBC_2.2) [SUSv3]	srandom(GLIBC_2.2) [SUSv3]	strtod(GLIBC_2.2) [SUSv3]
strtol(GLIBC_2.2) [SUSv3]	strtoul(GLIBC_2.2) [SUSv3]	swapcontext(GLIBC_2.2) [SUSv3]	syslog(GLIBC_2.2) [SUSv3]
system(GLIBC_2.2) [LSB]	tdelete(GLIBC_2.2) [SUSv3]	tfind(GLIBC_2.2) [SUSv3]	tmpfile(GLIBC_2.2) [SUSv3]
tmpnam(GLIBC_2.2) [SUSv3]	tsearch(GLIBC_2.2) [SUSv3]	ttyname(GLIBC_2.2) [SUSv3]	ttyname_r(GLIBC_2.2) [SUSv3]
twalk(GLIBC_2.2) [SUSv3]	unlockpt(GLIBC_2.2) [SUSv3]	unsetenv(GLIBC_2.2) [SUSv3]	usleep(GLIBC_2.2) [SUSv3]
verrx(GLIBC_2.2) [LSB]	vfscanf(GLIBC_2.2) [LSB]	vscanf(GLIBC_2.2) [LSB]	vsscanf(GLIBC_2.2) [LSB]
vsyslog(GLIBC_2.2) [LSB]	warn(GLIBC_2.2) [LSB]	warnx(GLIBC_2.2) [LSB]	wordexp(GLIBC_2.2.2) [SUSv3]
wordfree(GLIBC_2.2) [SUSv3]			

135

136

137

138

An LSB conforming implementation shall provide the architecture specific data interfaces for Standard Library specified in Table 11-23, with the full mandatory functionality as described in the referenced underlying specification.

139

**Table 11-23 libc - Standard Library Data Interfaces**

__environ(GLIBC_2.2) [LSB]	_environ(GLIBC_2.2) [LSB]	_sys_errlist(GLIBC_2.3) [LSB]	environ(GLIBC_2.2) [SUSv3]
getdate_err(GLIBC_2.2) [SUSv3]	optarg(GLIBC_2.2) [SUSv3]	opterr(GLIBC_2.2) [SUSv3]	optind(GLIBC_2.2) [SUSv3]
optopt(GLIBC_2.2) [SUSv3]			

140

### 11.3 Data Definitions for libc

141

142

143

144

This section defines global identifiers and their values that are associated with interfaces contained in libc. These definitions are organized into groups that correspond to system headers. This convention is used as a convenience for the reader, and does not imply the existence of these headers, or their content. Where an

145 interface is defined as requiring a particular system header file all of the data  
146 definitions for that system header file presented here shall be in effect.

147 This section gives data definitions to promote binary application portability, not to  
148 repeat source interface definitions available elsewhere. System providers and  
149 application developers should use this ABI to supplement - not to replace - source  
150 interface definition specifications.

151 This specification uses the ISO C (1999) C Language as the reference programming  
152 language, and data definitions are specified in ISO C format. The C language is used  
153 here as a convenient notation. Using a C language description of these data objects  
154 does not preclude their use by other programming languages.

### 11.3.1 arpa/inet.h

```
155 extern uint32_t htonl(uint32_t);
156 extern uint16_t htons(uint16_t);
157 extern in_addr_t inet_addr(const char *);
158 extern char *inet_ntoa(struct in_addr);
159 extern const char *inet_ntop(int, const void *, char *, socklen_t);
160 extern int inet_pton(int, const char *, void *);
161 extern uint32_t ntohl(uint32_t);
162 extern uint16_t ntohs(uint16_t);
163
```

### 11.3.2 assert.h

```
164
165 extern void __assert_fail(const char *, const char *, unsigned int,
166                          const char *);
```

### 11.3.3 ctype.h

```
167 extern int _tolower(int);
168 extern int _toupper(int);
169 extern int isalnum(int);
170 extern int isalpha(int);
171 extern int isascii(int);
172 extern int iscntrl(int);
173 extern int isdigit(int);
174 extern int isgraph(int);
175 extern int islower(int);
176 extern int isprint(int);
177 extern int ispunct(int);
178 extern int isspace(int);
179 extern int isupper(int);
180 extern int isxdigit(int);
181 extern int toascii(int);
182 extern int tolower(int);
183 extern int toupper(int);
184 extern int isblank(int);
185 extern const unsigned short **__ctype_b_loc(void);
186 extern const int32_t **__ctype_toupper_loc(void);
187 extern const int32_t **__ctype_tolower_loc(void);
188
```

### 11.3.4 dirent.h

```
189
190 extern void rewinddir(DIR *);
191 extern void seekdir(DIR *, long int);
192 extern long int telldir(DIR *);
```



```

193     extern int closedir(DIR *);
194     extern DIR *opendir(const char *);
195     extern struct dirent *readdir(DIR *);
196     extern struct dirent64 *readdir64(DIR *);
197     extern int readdir_r(DIR *, struct dirent *, struct dirent **);

```

### 11.3.5 err.h

```

198
199     extern void err(int, const char *, ...);
200     extern void errx(int, const char *, ...);
201     extern void warn(const char *, ...);
202     extern void warnx(const char *, ...);
203     extern void error(int, int, const char *, ...);

```

### 11.3.6 errno.h

```

204
205     #define EDEADLOCK          EDEADLK
206
207     extern int *__errno_location(void);

```

### 11.3.7 fcntl.h

```

208
209     #define F_GETLK64          5
210     #define F_SETLK64          6
211     #define F_SETLKW64        7
212
213     extern int lockf64(int, int, off64_t);
214     extern int fcntl(int, int, ...);

```

### 11.3.8 fmtmsg.h

```

215
216     extern int fmtmsg(long int, const char *, int, const char *, const char
217     *,
218                       const char *);

```

### 11.3.9 fnmatch.h

```

219
220     extern int fnmatch(const char *, const char *, int);

```

### 11.3.10 ftw.h

```

221
222     extern int ftw(const char *, __ftw_func_t, int);
223     extern int ftw64(const char *, __ftw64_func_t, int);
224     extern int nftw(const char *, __nftw_func_t, int, int);
225     extern int nftw64(const char *, __nftw64_func_t, int, int);

```

### 11.3.11 getopt.h

```

226
227     extern int getopt_long(int, char *const, const char *,
228                           const struct option *, int *);
229     extern int getopt_long_only(int, char *const, const char *,
230                                const struct option *, int *);

```

**11.3.12 glob.h**

```

231
232 extern int glob(const char *, int,
233                int (*__errfunc) (const char *p1, int p2)
234                , glob_t *);
235 extern int glob64(const char *, int,
236                  int (*__errfunc) (const char *p1, int p2)
237                  , glob64_t *);
238 extern void globfree(glob_t *);
239 extern void globfree64(glob64_t *);

```

**11.3.13 grp.h**

```

240
241 extern void endgrent(void);
242 extern struct group *getgrent(void);
243 extern struct group *getgrgid(gid_t);
244 extern struct group *getgrnam(char *);
245 extern int initgroups(const char *, gid_t);
246 extern void setgrent(void);
247 extern int setgroups(size_t, const gid_t *);
248 extern int getgrgid_r(gid_t, struct group *, char *, size_t,
249                      struct group **);
250 extern int getgrnam_r(const char *, struct group *, char *, size_t,
251                      struct group **);
252 extern int getgrouplist(const char *, gid_t, gid_t *, int *);

```

**11.3.14 iconv.h**

```

253
254 extern size_t iconv(iconv_t, char **, size_t *, char **, size_t *);
255 extern int iconv_close(iconv_t);
256 extern iconv_t iconv_open(char *, char *);

```

**11.3.15 inttypes.h**

```

257
258 typedef long int intmax_t;
259 typedef unsigned long int uintmax_t;
260 typedef unsigned long int uintptr_t;
261 typedef unsigned long int uint64_t;
262
263 extern intmax_t strtoumax(const char *, char **, int);
264 extern uintmax_t strtoumax(const char *, char **, int);
265 extern intmax_t wcstoumax(const wchar_t *, wchar_t * *, int);
266 extern uintmax_t wcstoumax(const wchar_t *, wchar_t * *, int);
267 extern intmax_t imaxabs(intmax_t);
268 extern imaxdiv_t imaxdiv(intmax_t, intmax_t);

```

**11.3.16 langinfo.h**

```

269
270 extern char *nl_langinfo(nl_item);

```

**11.3.17 libgen.h**

```

271
272 extern char *basename(const char *);
273 extern char *dirname(char *);

```

**11.3.18 libintl.h**

```

274
275 extern char *bindtextdomain(const char *, const char *);
276 extern char *dcgettext(const char *, const char *, int);
277 extern char *dgettext(const char *, const char *);
278 extern char *gettext(const char *);
279 extern char *textdomain(const char *);
280 extern char *bind_textdomain_codeset(const char *, const char *);
281 extern char *dcngettext(const char *, const char *, const char *,
282                        unsigned long int, int);
283 extern char *dngettext(const char *, const char *, const char *,
284                        unsigned long int);
285 extern char *ngettext(const char *, const char *, unsigned long int);

```

**11.3.19 limits.h**

```

286
287 #define LONG_MAX          0x7FFFFFFFFFFFFFFFL
288 #define ULONG_MAX        0xFFFFFFFFFFFFFFFFUL
289
290 #define CHAR_MAX          SCHAR_MAX
291 #define CHAR_MIN          SCHAR_MIN
292
293 #define PTHREAD_STACK_MIN 196608

```

**11.3.20 locale.h**

```

294
295 extern struct lconv *localeconv(void);
296 extern char *setlocale(int, const char *);
297 extern locale_t uselocale(locale_t);
298 extern void freelocale(locale_t);
299 extern locale_t duplocale(locale_t);
300 extern locale_t newlocale(int, const char *, locale_t);

```

**11.3.21 monetary.h**

```

301
302 extern ssize_t strfmon(char *, size_t, const char *, ...);

```

**11.3.22 net/if.h**

```

303
304 extern void if_freenameindex(struct if_nameindex *);
305 extern char *if_indextoname(unsigned int, char *);
306 extern struct if_nameindex *if_nameindex(void);
307 extern unsigned int if_nametoindex(const char *);

```

**11.3.23 netdb.h**

```

308
309 extern void endprotoent(void);
310 extern void endservent(void);
311 extern void freeaddrinfo(struct addrinfo *);
312 extern const char *gai_strerror(int);
313 extern int getaddrinfo(const char *, const char *, const struct addrinfo
314 *,
315                       struct addrinfo **);
316 extern struct hostent *gethostbyaddr(const void *, socklen_t, int);
317 extern struct hostent *gethostbyname(const char *);
318 extern struct protoent *getprotobyname(const char *);

```

```

319     extern struct protoent *getprotobynumber(int);
320     extern struct protoent *getprotoent(void);
321     extern struct servent *getservbyname(const char *, const char *);
322     extern struct servent *getservbyport(int, const char *);
323     extern struct servent *getservent(void);
324     extern void setprotoent(int);
325     extern void setservent(int);
326     extern int *__h_errno_location(void);

```

### 11.3.24 netinet/in.h

```

327
328     extern int bindresvport(int, struct sockaddr_in *);

```

### 11.3.25 netinet/ip.h

```

329
330     /*
331     * This header is architecture neutral
332     * Please refer to the generic specification for details
333     */

```

### 11.3.26 netinet/tcp.h

```

334
335     /*
336     * This header is architecture neutral
337     * Please refer to the generic specification for details
338     */

```

### 11.3.27 netinet/udp.h

```

339
340     /*
341     * This header is architecture neutral
342     * Please refer to the generic specification for details
343     */

```

### 11.3.28 nl\_types.h

```

344
345     extern int catclose(nl_catd);
346     extern char *catgets(nl_catd, int, int, const char *);
347     extern nl_catd catopen(const char *, int);

```

### 11.3.29 poll.h

```

348
349     extern int poll(struct pollfd *, nfd_t, int);

```

### 11.3.30 pty.h

```

350
351     extern int openpty(int *, int *, char *, struct termios *,
352                       struct winsize *);
353     extern int forkpty(int *, char *, struct termios *, struct winsize *);

```

### 11.3.31 pwd.h

```

354
355     extern void endpwent(void);
356     extern struct passwd *getpwent(void);

```

```

357 extern struct passwd *getpwnam(char *);
358 extern struct passwd *getpwuid(uid_t);
359 extern void setpwent(void);
360 extern int getpwnam_r(char *, struct passwd *, char *, size_t,
361                      struct passwd **);
362 extern int getpwuid_r(uid_t, struct passwd *, char *, size_t,
363                      struct passwd **);

```

### 11.3.32 regex.h

```

364
365 extern int regcomp(regex_t *, const char *, int);
366 extern size_t regerror(int, const regex_t *, char *, size_t);
367 extern int regexec(const regex_t *, const char *, size_t, regmatch_t,
368                  int);
369 extern void regfree(regex_t *);

```

### 11.3.33 rpc/auth.h

```

370
371 extern struct AUTH *authnone_create(void);
372 extern int key_decryptsession(char *, union des_block *);
373 extern bool_t xdr_opaque_auth(XDR *, struct opaque_auth *);

```

### 11.3.34 rpc/clnt.h

```

374
375 extern struct CLIENT *clnt_create(const char *, const u_long, const
376                                 u_long,
377                                 const char *);
378 extern void clnt_pcreateerror(const char *);
379 extern void clnt_perrno(enum clnt_stat);
380 extern void clnt_perror(struct CLIENT *, const char *);
381 extern char *clnt_screateerror(const char *);
382 extern char *clnt_serrno(enum clnt_stat);
383 extern char *clnt_serror(struct CLIENT *, const char *);

```

### 11.3.35 rpc/pmap\_clnt.h

```

384
385 extern u_short pmap_getport(struct sockaddr_in *, const u_long,
386                             const u_long, u_int);
387 extern bool_t pmap_set(const u_long, const u_long, int, u_short);
388 extern bool_t pmap_unset(u_long, u_long);

```

### 11.3.36 rpc/rpc\_msg.h

```

389
390 extern bool_t xdr_callhdr(XDR *, struct rpc_msg *);

```

### 11.3.37 rpc/svc.h

```

391
392 extern void svc_getreqset(fd_set *);
393 extern bool_t svc_register(SVCXPRT *, rpcprog_t, rpcvers_t,
394                           __dispatch_fn_t, rpcprot_t);
395 extern void svc_run(void);
396 extern bool_t svc_sendreply(SVCXPRT *, xdrproc_t, caddr_t);
397 extern void svcerr_auth(SVCXPRT *, enum auth_stat);
398 extern void svcerr_decode(SVCXPRT *);
399 extern void svcerr_noproc(SVCXPRT *);
400 extern void svcerr_noprogram(SVCXPRT *);

```

```

401 extern void svcerr_progvers(SVCXPRT *, rpcvers_t, rpcvers_t);
402 extern void svcerr_systemerr(SVCXPRT *);
403 extern void svcerr_weakauth(SVCXPRT *);
404 extern SVCXPRT *svctcp_create(int, u_int, u_int);
405 extern SVCXPRT *svcudp_create(int);

```

### 11.3.38 rpc/types.h

```

406
407 /*
408  * This header is architecture neutral
409  * Please refer to the generic specification for details
410  */

```

### 11.3.39 rpc/xdr.h

```

411
412 extern bool_t xdr_array(XDR *, caddr_t *, u_int *, u_int, u_int,
413                       xdrproc_t);
414 extern bool_t xdr_bool(XDR *, bool_t *);
415 extern bool_t xdr_bytes(XDR *, char **, u_int *, u_int);
416 extern bool_t xdr_char(XDR *, char *);
417 extern bool_t xdr_double(XDR *, double *);
418 extern bool_t xdr_enum(XDR *, enum_t *);
419 extern bool_t xdr_float(XDR *, float *);
420 extern void xdr_free(xdrproc_t, char *);
421 extern bool_t xdr_int(XDR *, int *);
422 extern bool_t xdr_long(XDR *, long int *);
423 extern bool_t xdr_opaque(XDR *, caddr_t, u_int);
424 extern bool_t xdr_pointer(XDR *, char **, u_int, xdrproc_t);
425 extern bool_t xdr_reference(XDR *, caddr_t *, u_int, xdrproc_t);
426 extern bool_t xdr_short(XDR *, short *);
427 extern bool_t xdr_string(XDR *, char **, u_int);
428 extern bool_t xdr_u_char(XDR *, u_char *);
429 extern bool_t xdr_u_int(XDR *, u_int *);
430 extern bool_t xdr_u_long(XDR *, u_long *);
431 extern bool_t xdr_u_short(XDR *, u_short *);
432 extern bool_t xdr_union(XDR *, enum_t *, char *,
433                       const struct xdr_discrim *, xdrproc_t);
434 extern bool_t xdr_vector(XDR *, char *, u_int, u_int, xdrproc_t);
435 extern bool_t xdr_void(void);
436 extern bool_t xdr_wrapstring(XDR *, char **);
437 extern void xdrmem_create(XDR *, caddr_t, u_int, enum xdr_op);
438 extern void xdrrec_create(XDR *, u_int, u_int, caddr_t,
439                          int (*__readit) (char *p1, char *p2, int p3)
440                          , int (*__writeit) (char *p1, char *p2, int
441                          p3)
442                          );
443 extern typedef int bool_t xdrrec_eof(XDR *);

```

### 11.3.40 sched.h

```

444
445 extern int sched_get_priority_max(int);
446 extern int sched_get_priority_min(int);
447 extern int sched_getparam(pid_t, struct sched_param *);
448 extern int sched_getscheduler(pid_t);
449 extern int sched_rr_get_interval(pid_t, struct timespec *);
450 extern int sched_setparam(pid_t, const struct sched_param *);
451 extern int sched_setscheduler(pid_t, int, const struct sched_param *);
452 extern int sched_yield(void);

```

**11.3.41 search.h**

```

453
454     extern int hcreate(size_t);
455     extern ENTRY *hsearch(ENTRY, ACTION);
456     extern void insque(void *, void *);
457     extern void *lfind(const void *, const void *, size_t *, size_t,
458                       __compar_fn_t);
459     extern void *lsearch(const void *, void *, size_t *, size_t,
460                         __compar_fn_t);
461     extern void remque(void *);
462     extern void hdestroy(void);
463     extern void *tdelete(const void *, void **, __compar_fn_t);
464     extern void *tfind(const void *, void *const *, __compar_fn_t);
465     extern void *tsearch(const void *, void **, __compar_fn_t);
466     extern void twalk(const void *, __action_fn_t);

```

**11.3.42 setjmp.h**

```

467
468     typedef long int __jmp_buf[70] __attribute__((aligned(16)));
469
470     extern int __sigsetjmp(jmp_buf, int);
471     extern void longjmp(jmp_buf, int);
472     extern void siglongjmp(sigjmp_buf, int);
473     extern void _longjmp(jmp_buf, int);
474     extern int _setjmp(jmp_buf);

```

**11.3.43 signal.h**

```

475
476     #define SIGEV_PAD_SIZE ((SIGEV_MAX_SIZE/sizeof(int))-4)
477
478     #define SI_PAD_SIZE ((SI_MAX_SIZE/sizeof(int))-4)
479
480     struct sigaction {
481         union {
482             sighandler_t _sa_handler;
483             void (*_sa_sigaction)(int, siginfo_t *, void *);
484         } __sigaction_handler;
485         unsigned long int sa_flags;
486         sigset_t sa_mask;
487     };
488
489     #define MINSIGSTKSZ 131027
490     #define SIGSTKSZ 262144
491
492     struct ia64_fpreg {
493         union {
494             unsigned long int bits[2];
495             long double __dummy;
496         } u;
497     };
498
499     struct sigcontext {
500         unsigned long int sc_flags;
501         unsigned long int sc_nat;
502         stack_t sc_stack;
503         unsigned long int sc_ip;
504         unsigned long int sc_cfm;
505         unsigned long int sc_um;
506         unsigned long int sc_ar_rsc;
507         unsigned long int sc_ar_bsp;

```

```

508         unsigned long int sc_ar_rnat;
509         unsigned long int sc_ar_ccv;
510         unsigned long int sc_ar_unat;
511         unsigned long int sc_ar_fpsr;
512         unsigned long int sc_ar_pfs;
513         unsigned long int sc_ar_lc;
514         unsigned long int sc_pr;
515         unsigned long int sc_br[8];
516         unsigned long int sc_gr[32];
517         struct ia64_fpreg sc_fr[128];
518         unsigned long int sc_rbs_base;
519         unsigned long int sc_loadrs;
520         unsigned long int sc_ar25;
521         unsigned long int sc_ar26;
522         unsigned long int sc_rsvd[12];
523         unsigned long int sc_mask;
524     };
525     extern int __libc_current_sigrtmax(void);
526     extern int __libc_current_sigrtmin(void);
527     extern sighandler_t __sysv_signal(int, sighandler_t);
528     extern char *const _sys_siglist(void);
529     extern int killpg(pid_t, int);
530     extern void psignal(int, const char *);
531     extern int raise(int);
532     extern int sigaddset(sigset_t *, int);
533     extern int sigandset(sigset_t *, const sigset_t *, const sigset_t *);
534     extern int sigdelset(sigset_t *, int);
535     extern int sigemptyset(sigset_t *);
536     extern int sigfillset(sigset_t *);
537     extern int sighold(int);
538     extern int sigignore(int);
539     extern int siginterrupt(int, int);
540     extern int sigisemptyset(const sigset_t *);
541     extern int sigismember(const sigset_t *, int);
542     extern int sigorset(sigset_t *, const sigset_t *, const sigset_t *);
543     extern int sigpending(sigset_t *);
544     extern int sigrelse(int);
545     extern sighandler_t sigset(int, sighandler_t);
546     extern int pthread_kill(pthread_t, int);
547     extern int pthread_sigmask(int, sigset_t *, sigset_t *);
548     extern int sigaction(int, const struct sigaction *, struct sigaction *);
549     extern int sigwait(sigset_t *, int *);
550     extern int kill(pid_t, int);
551     extern int sigaltstack(const struct sigaltstack *, struct sigaltstack
552     *);
553     extern sighandler_t signal(int, sighandler_t);
554     extern int sigpause(int);
555     extern int sigprocmask(int, const sigset_t *, sigset_t *);
556     extern int sigreturn(struct sigcontext *);
557     extern int sigsuspend(const sigset_t *);
558     extern int sigqueue(pid_t, int, const union sigval);
559     extern int sigwaitinfo(const sigset_t *, siginfo_t *);
560     extern int sigtimedwait(const sigset_t *, siginfo_t *,
561         const struct timespec *);
562     extern sighandler_t bsd_signal(int, sighandler_t);

```

### 11.3.44 stddef.h

```

563
564     typedef long int ptrdiff_t;
565     typedef unsigned long int size_t;

```



## 11.3.45 stdio.h

```

566
567 #define __IO_FILE_SIZE 216
568
569 extern char *const _sys_errlist(void);
570 extern void clearerr(FILE *);
571 extern int fclose(FILE *);
572 extern FILE *fdopen(int, const char *);
573 extern int fflush_unlocked(FILE *);
574 extern int fileno(FILE *);
575 extern FILE *fopen(const char *, const char *);
576 extern int fprintf(FILE *, const char *, ...);
577 extern int fputc(int, FILE *);
578 extern FILE *freopen(const char *, const char *, FILE *);
579 extern FILE *freopen64(const char *, const char *, FILE *);
580 extern int fscanf(FILE *, const char *, ...);
581 extern int fseek(FILE *, long int, int);
582 extern int fseeko(FILE *, off_t, int);
583 extern int fseeko64(FILE *, loff_t, int);
584 extern off_t ftello(FILE *);
585 extern loff_t ftello64(FILE *);
586 extern int getchar(void);
587 extern int getchar_unlocked(void);
588 extern int getw(FILE *);
589 extern int pclose(FILE *);
590 extern void perror(const char *);
591 extern FILE *popen(const char *, const char *);
592 extern int printf(const char *, ...);
593 extern int putc_unlocked(int, FILE *);
594 extern int putchar(int);
595 extern int putchar_unlocked(int);
596 extern int putw(int, FILE *);
597 extern int remove(const char *);
598 extern void rewind(FILE *);
599 extern int scanf(const char *, ...);
600 extern void setbuf(FILE *, char *);
601 extern int sprintf(char *, const char *, ...);
602 extern int sscanf(const char *, const char *, ...);
603 extern FILE *stderr(void);
604 extern FILE *stdin(void);
605 extern FILE *stdout(void);
606 extern char *tempnam(const char *, const char *);
607 extern FILE *tmpfile64(void);
608 extern FILE *tmpfile(void);
609 extern char *tmpnam(char *);
610 extern int vfprintf(FILE *, const char *, va_list);
611 extern int vprintf(const char *, va_list);
612 extern int feof(FILE *);
613 extern int ferror(FILE *);
614 extern int fflush(FILE *);
615 extern int fgetc(FILE *);
616 extern int fgetpos(FILE *, fpos_t *);
617 extern char *fgets(char *, int, FILE *);
618 extern int fputs(const char *, FILE *);
619 extern size_t fread(void *, size_t, size_t, FILE *);
620 extern int fsetpos(FILE *, const fpos_t *);
621 extern long int ftell(FILE *);
622 extern size_t fwrite(const void *, size_t, size_t, FILE *);
623 extern int getc(FILE *);
624 extern int putc(int, FILE *);
625 extern int puts(const char *);
626 extern int setvbuf(FILE *, char *, int, size_t);
627 extern int snprintf(char *, size_t, const char *, ...);

```

```

628     extern int ungetc(int, FILE *);
629     extern int vsnprintf(char *, size_t, const char *, va_list);
630     extern int vsprintf(char *, const char *, va_list);
631     extern void flockfile(FILE *);
632     extern int asprintf(char **, const char *, ...);
633     extern int fgetpos64(FILE *, fpos64_t *);
634     extern FILE *fopen64(const char *, const char *);
635     extern int fsetpos64(FILE *, const fpos64_t *);
636     extern int ftrylockfile(FILE *);
637     extern void funlockfile(FILE *);
638     extern int getc_unlocked(FILE *);
639     extern void setbuffer(FILE *, char *, size_t);
640     extern int vasprintf(char **, const char *, va_list);
641     extern int vdprintf(int, const char *, va_list);
642     extern int vfscanf(FILE *, const char *, va_list);
643     extern int vscanf(const char *, va_list);
644     extern int vsscanf(const char *, const char *, va_list);
645     extern size_t __fpending(FILE *);

```

### 11.3.46 stdlib.h

```

646
647     extern double __strtod_internal(const char *, char **, int);
648     extern float __strtof_internal(const char *, char **, int);
649     extern long int __strtoul_internal(const char *, char **, int, int);
650     extern long double __strtold_internal(const char *, char **, int);
651     extern long long int __strtoll_internal(const char *, char **, int, int);
652     extern unsigned long int __strtoul_internal(const char *, char **, int,
653                                               int);
654     extern unsigned long long int __strtoull_internal(const char *, char **,
655                                                      int, int);
656     extern long int a64l(const char *);
657     extern void abort(void);
658     extern int abs(int);
659     extern double atof(const char *);
660     extern int atoi(char *);
661     extern long int atol(char *);
662     extern long long int atoll(const char *);
663     extern void *bsearch(const void *, const void *, size_t, size_t,
664                          __compar_fn_t);
665     extern div_t div(int, int);
666     extern double drand48(void);
667     extern char *ecvt(double, int, int *, int *);
668     extern double erand48(unsigned short);
669     extern void exit(int);
670     extern char *fcvt(double, int, int *, int *);
671     extern char *gcvt(double, int, char *);
672     extern char *getenv(const char *);
673     extern int getsuopt(char **, char *const *, char **);
674     extern int grantpt(int);
675     extern long int jrand48(unsigned short);
676     extern char *l64a(long int);
677     extern long int labs(long int);
678     extern void lcong48(unsigned short);
679     extern ldiv_t ldiv(long int, long int);
680     extern long long int llabs(long long int);
681     extern lldiv_t lldiv(long long int, long long int);
682     extern long int lrand48(void);
683     extern int mblen(const char *, size_t);
684     extern size_t mbstowcs(wchar_t *, const char *, size_t);
685     extern int mbtowc(wchar_t *, const char *, size_t);
686     extern char *mktemp(char *);
687     extern long int mrand48(void);
688     extern long int nrand48(unsigned short);

```

```

689     extern char *ptsname(int);
690     extern int putenv(char *);
691     extern void qsort(void *, size_t, size_t, __compar_fn_t);
692     extern int rand(void);
693     extern int rand_r(unsigned int *);
694     extern unsigned short *seed48(unsigned short);
695     extern void srand48(long int);
696     extern int unlockpt(int);
697     extern size_t wcstombs(char *, const wchar_t *, size_t);
698     extern int wctomb(char *, wchar_t);
699     extern int system(const char *);
700     extern void *calloc(size_t, size_t);
701     extern void free(void *);
702     extern char *initstate(unsigned int, char *, size_t);
703     extern void *malloc(size_t);
704     extern long int random(void);
705     extern void *realloc(void *, size_t);
706     extern char *setstate(char *);
707     extern void srand(unsigned int);
708     extern void srandom(unsigned int);
709     extern double strtod(char *, char **);
710     extern float strtof(const char *, char **);
711     extern long int strtol(char *, char **, int);
712     extern long double strtold(const char *, char **);
713     extern long long int strtoll(const char *, char **, int);
714     extern long long int strtoll(const char *, char **, int);
715     extern unsigned long int strtoul(const char *, char **, int);
716     extern unsigned long long int strtoull(const char *, char **, int);
717     extern unsigned long long int strtoull(const char *, char **, int);
718     extern void _Exit(int);
719     extern size_t __ctype_get_mb_cur_max(void);
720     extern char **environ(void);
721     extern char *realpath(const char *, char *);
722     extern int setenv(const char *, const char *, int);
723     extern int unsetenv(const char *);
724     extern int getloadavg(double, int);
725     extern int mkstemp64(char *);
726     extern int posix_memalign(void **, size_t, size_t);
727     extern int posix_openpt(int);

```

### 11.3.47 string.h

```

728     extern void *__mempcpy(void *, const void *, size_t);
729     extern char *__stpcpy(char *, const char *);
730     extern char *__strtok_r(char *, const char *, char **);
731     extern void bcopy(void *, void *, size_t);
732     extern void *memchr(void *, int, size_t);
733     extern int memcmp(void *, void *, size_t);
734     extern void *memcpy(void *, void *, size_t);
735     extern void *memmem(const void *, size_t, const void *, size_t);
736     extern void *memmove(void *, const void *, size_t);
737     extern void *memset(void *, int, size_t);
738     extern char *strcat(char *, const char *);
739     extern char *strchr(char *, int);
740     extern int strcmp(char *, char *);
741     extern int strcoll(const char *, const char *);
742     extern char *strcpy(char *, char *);
743     extern size_t strcspn(const char *, const char *);
744     extern char *strerror(int);
745     extern size_t strlen(char *);
746     extern char *strncat(char *, char *, size_t);
747     extern int strncmp(char *, char *, size_t);
748     extern char *strncpy(char *, char *, size_t);
749

```

```

750     extern char *strpbrk(const char *, const char *);
751     extern char *strrchr(char *, int);
752     extern char *strsignal(int);
753     extern size_t strspn(const char *, const char *);
754     extern char *strstr(char *, char *);
755     extern char *strtok(char *, const char *);
756     extern size_t strxfrm(char *, const char *, size_t);
757     extern int bcmp(void *, void *, size_t);
758     extern void bzero(void *, size_t);
759     extern int ffs(int);
760     extern char *index(char *, int);
761     extern void *memccpy(void *, const void *, int, size_t);
762     extern char *rindex(char *, int);
763     extern int strcasecmp(char *, char *);
764     extern char *strdup(char *);
765     extern int strncasecmp(char *, char *, size_t);
766     extern char *strndup(const char *, size_t);
767     extern size_t strnlen(const char *, size_t);
768     extern char *strsep(char **, const char *);
769     extern char *strerror_r(int, char *, size_t);
770     extern char *strtok_r(char *, const char *, char **);
771     extern char *strcasestr(const char *, const char *);
772     extern char *stpncpy(char *, const char *);
773     extern char *stpcpy(char *, const char *, size_t);
774     extern void *memrchr(const void *, int, size_t);

```

### 11.3.48 sys/file.h

```

775
776     extern int flock(int, int);

```

### 11.3.49 sys/ioctl.h

```

777
778     #define TIOCGWINSZ      0x5413
779     #define FIONREAD       0x541B
780     #define TIOCNOTTY     0x5422
781
782     extern int ioctl(int, unsigned long int, ...);

```

### 11.3.50 sys/ipc.h

```

783
784     struct ipc_perm {
785         key_t __key;
786         uid_t uid;
787         gid_t gid;
788         uid_t cuid;
789         uid_t cgid;
790         mode_t mode;
791         unsigned short __seq;
792         unsigned short __pad1;
793         unsigned long int __unused1;
794         unsigned long int __unused2;
795     };
796
797     extern key_t ftok(char *, int);

```

### 11.3.51 sys/mman.h

```

798
799     #define MCL_CURRENT     1
800     #define MCL_FUTURE     2

```

```

801
802     extern int msync(void *, size_t, int);
803     extern int mlock(const void *, size_t);
804     extern int mlockall(int);
805     extern void *mmap(void *, size_t, int, int, int, off_t);
806     extern int mprotect(void *, size_t, int);
807     extern int munlock(const void *, size_t);
808     extern int munlockall(void);
809     extern int munmap(void *, size_t);
810     extern void *mmap64(void *, size_t, int, int, int, off64_t);
811     extern int shm_open(const char *, int, mode_t);
812     extern int shm_unlink(const char *);

```

### 11.3.52 sys/msg.h

```

813
814     struct msqid_ds {
815         struct ipc_perm msg_perm;
816         time_t msg_stime;
817         time_t msg_rtime;
818         time_t msg_ctime;
819         unsigned long int __msg_cbytes;
820         unsigned long int msg_qnum;
821         unsigned long int msg_qbytes;
822         pid_t msg_lspid;
823         pid_t msg_lrpid;
824         unsigned long int __unused1;
825         unsigned long int __unused2;
826     };
827     extern int msgctl(int, int, struct msqid_ds *);
828     extern int msgget(key_t, int);
829     extern int msgrcv(int, void *, size_t, long int, int);
830     extern int msgsnd(int, const void *, size_t, int);

```

### 11.3.53 sys/param.h

```

831
832     /*
833     * This header is architecture neutral
834     * Please refer to the generic specification for details
835     */

```

### 11.3.54 sys/poll.h

```

836
837     /*
838     * This header is architecture neutral
839     * Please refer to the generic specification for details
840     */

```

### 11.3.55 sys/resource.h

```

841
842     extern int getpriority(__priority_which_t, id_t);
843     extern int getrlimit64(id_t, struct rlimit64 *);
844     extern int setpriority(__priority_which_t, id_t, int);
845     extern int setrlimit(__rlimit_resource_t, const struct rlimit *);
846     extern int setrlimit64(__rlimit_resource_t, const struct rlimit64 *);
847     extern int getrlimit(__rlimit_resource_t, struct rlimit *);
848     extern int getrusage(int, struct rusage *);

```

**11.3.56 sys/sem.h**

```

849
850 struct semid_ds {
851     struct ipc_perm sem_perm;
852     time_t sem_otime;
853     time_t sem_ctime;
854     unsigned long int sem_nsems;
855     unsigned long int __unused1;
856     unsigned long int __unused2;
857 };
858 extern int semctl(int, int, int, ...);
859 extern int semget(key_t, int, int);
860 extern int semop(int, struct sembuf *, size_t);

```

**11.3.57 sys/shm.h**

```

861
862 #define SHMLBA (1024*1024)
863
864 struct shmid_ds {
865     struct ipc_perm shm_perm;
866     size_t shm_segsz;
867     time_t shm_atime;
868     time_t shm_dtime;
869     time_t shm_ctime;
870     pid_t shm_cpid;
871     pid_t shm_lpid;
872     unsigned long int shm_nattch;
873     unsigned long int __unused1;
874     unsigned long int __unused2;
875 };
876 extern int __getpagesize(void);
877 extern void *shmat(int, const void *, int);
878 extern int shmctl(int, int, struct shmid_ds *);
879 extern int shmdt(const void *);
880 extern int shmget(key_t, size_t, int);

```

**11.3.58 sys/socket.h**

```

881
882 typedef uint64_t __ss_aligntype;
883
884 #define SO_RCVLOWAT 18
885 #define SO_SNDLOWAT 19
886 #define SO_RCVTIMEO 20
887 #define SO_SNDTIMEO 21
888
889 extern int bind(int, const struct sockaddr *, socklen_t);
890 extern int getnameinfo(const struct sockaddr *, socklen_t, char *,
891                       socklen_t, char *, socklen_t, unsigned int);
892 extern int getsockname(int, struct sockaddr *, socklen_t *);
893 extern int listen(int, int);
894 extern int setsockopt(int, int, int, const void *, socklen_t);
895 extern int accept(int, struct sockaddr *, socklen_t *);
896 extern int connect(int, const struct sockaddr *, socklen_t);
897 extern ssize_t recv(int, void *, size_t, int);
898 extern ssize_t recvfrom(int, void *, size_t, int, struct sockaddr *,
899                        socklen_t *);
900 extern ssize_t recvmsg(int, struct msghdr *, int);
901 extern ssize_t send(int, const void *, size_t, int);
902 extern ssize_t sendmsg(int, const struct msghdr *, int);
903 extern ssize_t sendto(int, const void *, size_t, int,

```

```

904             const struct sockaddr *, socklen_t);
905 extern int getpeername(int, struct sockaddr *, socklen_t *);
906 extern int getsockopt(int, int, int, void *, socklen_t *);
907 extern int shutdown(int, int);
908 extern int socket(int, int, int);
909 extern int socketpair(int, int, int, int);
910 extern int socketatmark(int);

```

### 11.3.59 sys/stat.h

```

911
912 #define _STAT_VER      1
913
914 struct stat {
915     dev_t st_dev;
916     ino_t st_ino;
917     nlink_t st_nlink;
918     mode_t st_mode;
919     uid_t st_uid;
920     gid_t st_gid;
921     unsigned int pad0;
922     dev_t st_rdev;
923     off_t st_size;
924     struct timespec st_atim;
925     struct timespec st_mtim;
926     struct timespec st_ctim;
927     blksize_t st_blksize;
928     blkcnt_t st_blocks;
929     unsigned long int __unused[3];
930 };
931 struct stat64 {
932     dev_t st_dev;
933     ino64_t st_ino;
934     nlink_t st_nlink;
935     mode_t st_mode;
936     uid_t st_uid;
937     gid_t st_gid;
938     unsigned int pad0;
939     dev_t st_rdev;
940     off_t st_size;
941     struct timespec st_atim;
942     struct timespec st_mtim;
943     struct timespec st_ctim;
944     blksize_t st_blksize;
945     blkcnt64_t st_blocks;
946     unsigned long int __unused[3];
947 };
948
949 extern int __fxstat(int, int, struct stat *);
950 extern int __fxstat64(int, int, struct stat64 *);
951 extern int __lxstat(int, char *, struct stat *);
952 extern int __lxstat64(int, const char *, struct stat64 *);
953 extern int __xmknod(int, const char *, mode_t, dev_t *);
954 extern int __xstat(int, const char *, struct stat *);
955 extern int __xstat64(int, const char *, struct stat64 *);
956 extern int mkfifo(const char *, mode_t);
957 extern int chmod(const char *, mode_t);
958 extern int fchmod(int, mode_t);
959 extern mode_t umask(mode_t);

```

### 11.3.60 sys/statvfs.h

```

960
961 struct statvfs {

```

```

962         unsigned long int f_bsize;
963         unsigned long int f_frsize;
964         fsblkcnt64_t f_blocks;
965         fsblkcnt64_t f_bfree;
966         fsblkcnt64_t f_bavail;
967         fsfilcnt64_t f_files;
968         fsfilcnt64_t f_ffree;
969         fsfilcnt64_t f_favail;
970         unsigned long int f_fsid;
971         unsigned long int f_flag;
972         unsigned long int f_namemax;
973         unsigned int __f_spare[6];
974     };
975     struct statvfs64 {
976         unsigned long int f_bsize;
977         unsigned long int f_frsize;
978         fsblkcnt64_t f_blocks;
979         fsblkcnt64_t f_bfree;
980         fsblkcnt64_t f_bavail;
981         fsfilcnt64_t f_files;
982         fsfilcnt64_t f_ffree;
983         fsfilcnt64_t f_favail;
984         unsigned long int f_fsid;
985         unsigned long int f_flag;
986         unsigned long int f_namemax;
987         unsigned int __f_spare[6];
988     };
989     extern int fstatvfs(int, struct statvfs *);
990     extern int fstatvfs64(int, struct statvfs64 *);
991     extern int statvfs(const char *, struct statvfs *);
992     extern int statvfs64(const char *, struct statvfs64 *);

```

### 11.3.61 sys/time.h

```

993
994     extern int getitimer(__itimer_which_t, struct itimerval *);
995     extern int setitimer(__itimer_which_t, const struct itimerval *,
996                         struct itimerval *);
997     extern int adjtime(const struct timeval *, struct timeval *);
998     extern int gettimeofday(struct timeval *, struct timezone *);
999     extern int utimes(const char *, const struct timeval *);

```

### 11.3.62 sys/timeb.h

```

1000
1001     extern int ftime(struct timeb *);

```

### 11.3.63 sys/times.h

```

1002
1003     extern clock_t times(struct tms *);

```

### 11.3.64 sys/types.h

```

1004
1005     typedef long int int64_t;
1006
1007     typedef int64_t ssize_t;
1008
1009     #define __FDSET_LONGS 16

```



**11.3.65 sys/uio.h**

```

1010
1011 extern ssize_t readv(int, const struct iovec *, int);
1012 extern ssize_t writev(int, const struct iovec *, int);

```

**11.3.66 sys/un.h**

```

1013
1014 /*
1015  * This header is architecture neutral
1016  * Please refer to the generic specification for details
1017  */

```

**11.3.67 sys/utsname.h**

```

1018
1019 extern int uname(struct utsname *);

```

**11.3.68 sys/wait.h**

```

1020
1021 extern pid_t wait(int *);
1022 extern pid_t waitpid(pid_t, int *, int);
1023 extern pid_t wait4(pid_t, int *, int, struct rusage *);

```

**11.3.69 syslog.h**

```

1024
1025 extern void closelog(void);
1026 extern void openlog(const char *, int, int);
1027 extern int setlogmask(int);
1028 extern void syslog(int, const char *, ...);
1029 extern void vsyslog(int, const char *, va_list);

```

**11.3.70 termios.h**

```

1030
1031 #define OLCUC    0000002
1032 #define ONLCR   0000004
1033 #define XCASE   0000004
1034 #define NLDLY   0000400
1035 #define CR1     0001000
1036 #define IUCLC   0001000
1037 #define CR2     0002000
1038 #define CR3     0003000
1039 #define CRDLY   0003000
1040 #define TAB1    0004000
1041 #define TAB2    0010000
1042 #define TAB3    0014000
1043 #define TABDLY  0014000
1044 #define BS1     0020000
1045 #define BSDLY   0020000
1046 #define VT1     0040000
1047 #define VTDLY   0040000
1048 #define FF1     0100000
1049 #define FFDLY   0100000
1050
1051 #define VSUSP   10
1052 #define VEOL    11
1053 #define VREPRINT 12
1054 #define VDISCARD 13

```

## 11 Libraries

```
1055     #define VWERASE 14
1056     #define VEOL2   16
1057     #define VMIN    6
1058     #define VSWTC   7
1059     #define VSTART  8
1060     #define VSTOP   9
1061
1062     #define IXON    0002000
1063     #define IXOFF  0010000
1064
1065     #define CS6     0000020
1066     #define CS7     0000040
1067     #define CS8     0000060
1068     #define CSIZE   0000060
1069     #define CSTOPB  0000100
1070     #define CREAD   0000200
1071     #define PARENB  0000400
1072     #define PARODD  0001000
1073     #define HUPCL   0002000
1074     #define CLOCAL  0004000
1075     #define VTIME   5
1076
1077     #define ISIG    0000001
1078     #define ICANON  0000002
1079     #define ECHOE   0000020
1080     #define ECHOK   0000040
1081     #define ECHONL  0000100
1082     #define NOFLSH  0000200
1083     #define TOSTOP  0000400
1084     #define ECHOCTL 0001000
1085     #define ECHOPRT 0002000
1086     #define ECHOKE  0004000
1087     #define FLUSHO  0010000
1088     #define PENDIN  0040000
1089     #define IEXTEN  0100000
1090
1091     extern speed_t cfgetispeed(const struct termios *);
1092     extern speed_t cfgetospeed(const struct termios *);
1093     extern void cfmakeraw(struct termios *);
1094     extern int cfsetispeed(struct termios *, speed_t);
1095     extern int cfsetospeed(struct termios *, speed_t);
1096     extern int cfsetspeed(struct termios *, speed_t);
1097     extern int tcflow(int, int);
1098     extern int tcflush(int, int);
1099     extern pid_t tcgetsid(int);
1100     extern int tcsendbreak(int, int);
1101     extern int tcsetattr(int, int, const struct termios *);
1102     extern int tcdrain(int);
1103     extern int tcgetattr(int, struct termios *);
```

### 11.3.71 time.h

```
1104
1105     extern int __daylight(void);
1106     extern long int __timezone(void);
1107     extern char *__tzname(void);
1108     extern char *asctime(const struct tm *);
1109     extern clock_t clock(void);
1110     extern char *ctime(const time_t *);
1111     extern char *ctime_r(const time_t *, char *);
1112     extern double difftime(time_t, time_t);
1113     extern struct tm *getdate(const char *);
1114     extern int getdate_err(void);
1115     extern struct tm *gmtime(const time_t *);
```

```

1116 extern struct tm *localtime(const time_t *);
1117 extern time_t mktime(struct tm *);
1118 extern int stime(const time_t *);
1119 extern size_t strftime(char *, size_t, const char *, const struct tm *);
1120 extern char *strptime(const char *, const char *, struct tm *);
1121 extern time_t time(time_t *);
1122 extern int nanosleep(const struct timespec *, struct timespec *);
1123 extern int daylight(void);
1124 extern long int timezone(void);
1125 extern char *tzname(void);
1126 extern void tzset(void);
1127 extern char *asctime_r(const struct tm *, char *);
1128 extern struct tm *gmtime_r(const time_t *, struct tm *);
1129 extern struct tm *localtime_r(const time_t *, struct tm *);
1130 extern int clock_getcpuclockid(pid_t, clockid_t *);
1131 extern int clock_getres(clockid_t, struct timespec *);
1132 extern int clock_gettime(clockid_t, struct timespec *);
1133 extern int clock_nanosleep(clockid_t, int, const struct timespec *,
1134                          struct timespec *);
1135 extern int clock_settime(clockid_t, const struct timespec *);
1136 extern int timer_create(clockid_t, struct sigevent *, timer_t *);
1137 extern int timer_delete(timer_t);
1138 extern int timer_getoverrun(timer_t);
1139 extern int timer_gettime(timer_t, struct itimerspec *);
1140 extern int timer_settime(timer_t, int, const struct itimerspec *,
1141                          struct itimerspec *);

```

### 11.3.72 ucontext.h

```

1142
1143 #define _SC_GRO_OFFSET \
1144     (((char *) &((struct sigcontext *) 0)->sc_gr[0]) - (char *) 0)
1145
1146 typedef struct sigcontext mcontext_t;
1147
1148 typedef struct ucontext {
1149     union {
1150         mcontext_t _mc;
1151         struct {
1152             unsigned long int _pad[_SC_GRO_OFFSET / 8];
1153             struct ucontext *_link;
1154         } _uc;
1155     } _u;
1156 } ucontext_t;
1157 extern int getcontext(ucontext_t *);
1158 extern int makecontext(ucontext_t *, void (*func) (void)
1159                      , int, ...);
1160 extern int setcontext(const struct ucontext *);
1161 extern int swapcontext(ucontext_t *, const struct ucontext *);

```

### 11.3.73 ulimit.h

```

1162
1163 extern long int ulimit(int, ...);

```

### 11.3.74 unistd.h

```

1164
1165 typedef long int intptr_t;
1166
1167 extern char **__environ(void);
1168 extern pid_t __getpgid(pid_t);
1169 extern void _exit(int);

```

```

1170     extern int acct(const char *);
1171     extern unsigned int alarm(unsigned int);
1172     extern int chown(const char *, uid_t, gid_t);
1173     extern int chroot(const char *);
1174     extern size_t confstr(int, char *, size_t);
1175     extern int creat(const char *, mode_t);
1176     extern int creat64(const char *, mode_t);
1177     extern char *ctermid(char *);
1178     extern char *cuserid(char *);
1179     extern int daemon(int, int);
1180     extern int execl(const char *, const char *, ...);
1181     extern int execlp(const char *, const char *, ...);
1182     extern int execlp(const char *, const char *, ...);
1183     extern int execv(const char *, char *const);
1184     extern int execvp(const char *, char *const);
1185     extern int fdatsync(int);
1186     extern int ftruncate64(int, off64_t);
1187     extern long int gethostid(void);
1188     extern char *getlogin(void);
1189     extern int getlogin_r(char *, size_t);
1190     extern int getopt(int, char *const, const char *);
1191     extern pid_t getpgrp(void);
1192     extern pid_t getsid(pid_t);
1193     extern char *getwd(char *);
1194     extern int lockf(int, int, off_t);
1195     extern int mkstemp(char *);
1196     extern int nice(int);
1197     extern char *optarg(void);
1198     extern int opterr(void);
1199     extern int optind(void);
1200     extern int optopt(void);
1201     extern int rename(const char *, const char *);
1202     extern int setegid(gid_t);
1203     extern int seteuid(uid_t);
1204     extern int sethostname(const char *, size_t);
1205     extern int setpgrp(void);
1206     extern void swab(const void *, void *, ssize_t);
1207     extern void sync(void);
1208     extern pid_t tcgetpgrp(int);
1209     extern int tcsetpgrp(int, pid_t);
1210     extern int truncate(const char *, off_t);
1211     extern int truncate64(const char *, off64_t);
1212     extern char *ttyname(int);
1213     extern unsigned int ualarm(useconds_t, useconds_t);
1214     extern int usleep(useconds_t);
1215     extern int close(int);
1216     extern int fsync(int);
1217     extern off_t lseek(int, off_t, int);
1218     extern int open(const char *, int, ...);
1219     extern int pause(void);
1220     extern ssize_t read(int, void *, size_t);
1221     extern ssize_t write(int, const void *, size_t);
1222     extern char *crypt(char *, char *);
1223     extern void encrypt(char *, int);
1224     extern void setkey(const char *);
1225     extern int access(const char *, int);
1226     extern int brk(void *);
1227     extern int chdir(const char *);
1228     extern int dup(int);
1229     extern int dup2(int, int);
1230     extern int execve(const char *, char *const, char *const);
1231     extern int fchdir(int);
1232     extern int fchown(int, uid_t, gid_t);
1233     extern pid_t fork(void);

```

```

1234     extern gid_t getegid(void);
1235     extern uid_t geteuid(void);
1236     extern gid_t getgid(void);
1237     extern int getgroups(int, gid_t);
1238     extern int gethostname(char *, size_t);
1239     extern pid_t getpgid(pid_t);
1240     extern pid_t getpid(void);
1241     extern uid_t getuid(void);
1242     extern int lchown(const char *, uid_t, gid_t);
1243     extern int link(const char *, const char *);
1244     extern int mkdir(const char *, mode_t);
1245     extern long int pathconf(const char *, int);
1246     extern int pipe(int);
1247     extern int readlink(const char *, char *, size_t);
1248     extern int rmdir(const char *);
1249     extern void *sbrk(ptrdiff_t);
1250     extern int select(int, fd_set *, fd_set *, fd_set *, struct timeval *);
1251     extern int setgid(gid_t);
1252     extern int setpgid(pid_t, pid_t);
1253     extern int setregid(gid_t, gid_t);
1254     extern int setreuid(uid_t, uid_t);
1255     extern pid_t setsid(void);
1256     extern int setuid(uid_t);
1257     extern unsigned int sleep(unsigned int);
1258     extern int symlink(const char *, const char *);
1259     extern long int sysconf(int);
1260     extern int unlink(const char *);
1261     extern pid_t vfork(void);
1262     extern ssize_t pread(int, void *, size_t, off_t);
1263     extern ssize_t pwrite(int, const void *, size_t, off_t);
1264     extern char **_environ(void);
1265     extern long int fpathconf(int, int);
1266     extern int ftruncate(int, off_t);
1267     extern char *getcwd(char *, size_t);
1268     extern int getpagesize(void);
1269     extern pid_t getppid(void);
1270     extern int isatty(int);
1271     extern loff_t lseek64(int, loff_t, int);
1272     extern int open64(const char *, int, ...);
1273     extern ssize_t pread64(int, void *, size_t, off64_t);
1274     extern ssize_t pwrite64(int, const void *, size_t, off64_t);
1275     extern int ttyname_r(int, char *, size_t);

```

### 11.3.75 utime.h

```

1276     extern int utime(const char *, const struct utimbuf *);
1277

```

### 11.3.76 utmp.h

```

1278     struct lastlog {
1279         time_t ll_time;
1280         char ll_line[UT_LINESIZE];
1281         char ll_host[UT_HOSTSIZE];
1282     };
1283
1284     struct utmp {
1285         short ut_type;
1286         pid_t ut_pid;
1287         char ut_line[UT_LINESIZE];
1288         char ut_id[4];
1289         char ut_user[UT_NAMESIZE];
1290         char ut_host[UT_HOSTSIZE];
1291

```

```

1292         struct exit_status ut_exit;
1293         long int ut_session;
1294         struct timeval ut_tv;
1295         int32_t ut_addr_v6[4];
1296         char __unused[20];
1297     };
1298
1299     extern void endutent(void);
1300     extern struct utmp *getutent(void);
1301     extern void setutent(void);
1302     extern int getutent_r(struct utmp *, struct utmp **);
1303     extern int utmpname(const char *);
1304     extern int login_tty(int);
1305     extern void login(const struct utmp *);
1306     extern int logout(const char *);
1307     extern void logwtmp(const char *, const char *, const char *);

```

### 11.3.77 utmpx.h

```

1308
1309     struct utmpx {
1310         short ut_type;
1311         pid_t ut_pid;
1312         char ut_line[UT_LINESIZE];
1313         char ut_id[4];
1314         char ut_user[UT_NAMESIZE];
1315         char ut_host[UT_HOSTSIZE];
1316         struct exit_status ut_exit;
1317         long int ut_session;
1318         struct timeval ut_tv;
1319         int32_t ut_addr_v6[4];
1320         char __unused[20];
1321     };
1322
1323     extern void endutxent(void);
1324     extern struct utmpx *getutxent(void);
1325     extern struct utmpx *getutxid(const struct utmpx *);
1326     extern struct utmpx *getutxline(const struct utmpx *);
1327     extern struct utmpx *pututxline(const struct utmpx *);
1328     extern void setutxent(void);

```

### 11.3.78 wchar.h

```

1329
1330     extern double __wcstod_internal(const wchar_t *, wchar_t **, int);
1331     extern float __wcstof_internal(const wchar_t *, wchar_t **, int);
1332     extern long int __wcstol_internal(const wchar_t *, wchar_t **, int,
1333     int);
1334     extern long double __wcstold_internal(const wchar_t *, wchar_t **, int);
1335     extern unsigned long int __wcstoul_internal(const wchar_t *, wchar_t *
1336     *,
1337     int, int);
1338     extern wchar_t *wcscat(wchar_t *, const wchar_t *);
1339     extern wchar_t *wcschr(const wchar_t *, wchar_t);
1340     extern int wcsncmp(const wchar_t *, const wchar_t *);
1341     extern int wscoll(const wchar_t *, const wchar_t *);
1342     extern wchar_t *wcscpy(wchar_t *, const wchar_t *);
1343     extern size_t wcsncpy(const wchar_t *, const wchar_t *);
1344     extern wchar_t *wcsdup(const wchar_t *);
1345     extern wchar_t *wcsncat(wchar_t *, const wchar_t *, size_t);
1346     extern int wcsncmp(const wchar_t *, const wchar_t *, size_t);
1347     extern wchar_t *wcsncpy(wchar_t *, const wchar_t *, size_t);
1348     extern wchar_t *wcpbrk(const wchar_t *, const wchar_t *);
1349     extern wchar_t *wcsrchr(const wchar_t *, wchar_t);

```

```

1350 extern size_t wcsnlen(const wchar_t *, size_t);
1351 extern wchar_t *wcsstr(const wchar_t *, const wchar_t *);
1352 extern wchar_t *wcstok(wchar_t *, const wchar_t *, wchar_t * *);
1353 extern int wcswidth(const wchar_t *, size_t);
1354 extern size_t wcsxfrm(wchar_t *, const wchar_t *, size_t);
1355 extern int wctob(wint_t);
1356 extern int wcwidth(wchar_t);
1357 extern wchar_t *wmemchr(const wchar_t *, wchar_t, size_t);
1358 extern int wmemcmp(const wchar_t *, const wchar_t *, size_t);
1359 extern wchar_t *wmemcpy(wchar_t *, const wchar_t *, size_t);
1360 extern wchar_t *wmemmove(wchar_t *, const wchar_t *, size_t);
1361 extern wchar_t *wmemset(wchar_t *, wchar_t, size_t);
1362 extern size_t mbrlen(const char *, size_t, mbstate_t *);
1363 extern size_t mbrtowc(wchar_t *, const char *, size_t, mbstate_t *);
1364 extern int mbsinit(const mbstate_t *);
1365 extern size_t mbsnrtowcs(wchar_t *, const char **, size_t, size_t,
1366     mbstate_t *);
1367 extern size_t mbsrtowcs(wchar_t *, const char **, size_t, mbstate_t *);
1368 extern wchar_t *wcpcpy(wchar_t *, const wchar_t *);
1369 extern wchar_t *wcpncpy(wchar_t *, const wchar_t *, size_t);
1370 extern size_t wcrntomb(char *, wchar_t, mbstate_t *);
1371 extern size_t wcslen(const wchar_t *);
1372 extern size_t wcsnrtombs(char *, const wchar_t * *, size_t, size_t,
1373     mbstate_t *);
1374 extern size_t wcsrtombs(char *, const wchar_t * *, size_t, mbstate_t *);
1375 extern double wcstod(const wchar_t *, wchar_t * *);
1376 extern float wcstof(const wchar_t *, wchar_t * *);
1377 extern long int wcstol(const wchar_t *, wchar_t * *, int);
1378 extern long double wcstold(const wchar_t *, wchar_t * *);
1379 extern long long int wcstoq(const wchar_t *, wchar_t * *, int);
1380 extern unsigned long int wcstoul(const wchar_t *, wchar_t * *, int);
1381 extern unsigned long long int wcstoull(const wchar_t *, wchar_t * *, int);
1382 extern wchar_t *wswcs(const wchar_t *, const wchar_t *);
1383 extern int wscasecmp(const wchar_t *, const wchar_t *);
1384 extern int wcsncasecmp(const wchar_t *, const wchar_t *, size_t);
1385 extern size_t wcsnlen(const wchar_t *, size_t);
1386 extern long long int wcstoll(const wchar_t *, wchar_t * *, int);
1387 extern unsigned long long int wcstoull(const wchar_t *, wchar_t * *, int);
1388 extern wint_t btowc(int);
1389 extern wint_t fgetwc(FILE *);
1390 extern wint_t fgetwc_unlocked(FILE *);
1391 extern wchar_t *fgetws(wchar_t *, int, FILE *);
1392 extern wint_t fputwc(wchar_t, FILE *);
1393 extern int fputws(const wchar_t *, FILE *);
1394 extern int fwide(FILE *, int);
1395 extern int fwprintf(FILE *, const wchar_t *, ...);
1396 extern int fwscanf(FILE *, const wchar_t *, ...);
1397 extern wint_t getwc(FILE *);
1398 extern wint_t getwchar(void);
1399 extern wint_t putwc(wchar_t, FILE *);
1400 extern wint_t putwchar(wchar_t);
1401 extern int swprintf(wchar_t *, size_t, const wchar_t *, ...);
1402 extern int swscanf(const wchar_t *, const wchar_t *, ...);
1403 extern wint_t ungetwc(wint_t, FILE *);
1404 extern int vfwprintf(FILE *, const wchar_t *, va_list);
1405 extern int vfwscanf(FILE *, const wchar_t *, va_list);
1406 extern int vswprintf(wchar_t *, size_t, const wchar_t *, va_list);
1407 extern int vswscanf(const wchar_t *, const wchar_t *, va_list);
1408 extern int vwprintf(const wchar_t *, va_list);
1409 extern int vwscanf(const wchar_t *, va_list);
1410 extern size_t wcsftime(wchar_t *, size_t, const wchar_t *,
1411     const struct tm *);
1412 extern int wprintf(const wchar_t *, ...);
1413 extern int wscanf(const wchar_t *, ...);

```

**11.3.79 wctype.h**

```

1414
1415     extern int iswblank(wint_t);
1416     extern wint_t towlower(wint_t);
1417     extern wint_t towupper(wint_t);
1418     extern wctrans_t wctrans(const char *);
1419     extern int iswalnum(wint_t);
1420     extern int iswalphabetic(wint_t);
1421     extern int iswcntrl(wint_t);
1422     extern int iswctype(wint_t, wctype_t);
1423     extern int iswdigit(wint_t);
1424     extern int iswgraph(wint_t);
1425     extern int iswlower(wint_t);
1426     extern int iswprint(wint_t);
1427     extern int iswpunct(wint_t);
1428     extern int iswspace(wint_t);
1429     extern int iswupper(wint_t);
1430     extern int iswxdigit(wint_t);
1431     extern wctype_t wctype(const char *);
1432     extern wint_t towctrans(wint_t, wctrans_t);

```

**11.3.80 wordexp.h**

```

1433
1434     extern int wordexp(const char *, wordexp_t *, int);
1435     extern void wordfree(wordexp_t *);

```

**11.4 Interfaces for libm**

1436 Table 11-24 defines the library name and shared object name for the libm library

1437 **Table 11-24 libm Definition**

Library:	libm
SONAME:	libm.so.6.1

1438  
1439 The behavior of the interfaces in this library is specified by the following specifica-  
1440 tions:

1441 [ISOC99] ISO C (1999)  
[LSB] This Specification  
[SUSv2] SUSv2  
[SUSv3] ISO POSIX (2003)

**11.4.1 Math****11.4.1.1 Interfaces for Math**

1442  
1443 An LSB conforming implementation shall provide the architecture specific functions  
1444 for Math specified in Table 11-25, with the full mandatory functionality as described  
1445 in the referenced underlying specification.

1446 **Table 11-25 libm - Math Function Interfaces**

__finite(GLIBC_2.2) [ISOC99]	__finitef(GLIBC_2.2) [ISOC99]	__finitel(GLIBC_2.2) [ISOC99]	__fpclassify(GLIBC_2.2) [LSB]
__fpclassifyf(GLIBC_2.2)	__fpclassifyl(GLIBC_2.2)	__signbit(GLIBC_2.2)	__signbitf(GLIBC_2.2)



C_2.2) [LSB]	C_2.2) [ISOC99]	2.2) [ISOC99]	_2.2) [ISOC99]
__signbitl(GLIBC_2.2) [ISOC99]	acos(GLIBC_2.2) [SUSv3]	acosf(GLIBC_2.2) [SUSv3]	acosh(GLIBC_2.2) [SUSv3]
acoshf(GLIBC_2.2) [SUSv3]	acoshl(GLIBC_2.2) [SUSv3]	acosl(GLIBC_2.2) [SUSv3]	asin(GLIBC_2.2) [SUSv3]
asinf(GLIBC_2.2) [SUSv3]	asinh(GLIBC_2.2) [SUSv3]	asinhf(GLIBC_2.2) [SUSv3]	asinhL(GLIBC_2.2) [SUSv3]
asinl(GLIBC_2.2) [SUSv3]	atan(GLIBC_2.2) [SUSv3]	atan2(GLIBC_2.2) [SUSv3]	atan2f(GLIBC_2.2) [SUSv3]
atan2l(GLIBC_2.2) [SUSv3]	atanf(GLIBC_2.2) [SUSv3]	atanh(GLIBC_2.2) [SUSv3]	atanhf(GLIBC_2.2) [SUSv3]
atanhl(GLIBC_2.2) [SUSv3]	atanl(GLIBC_2.2) [SUSv3]	cabs(GLIBC_2.2) [SUSv3]	cabsf(GLIBC_2.2) [SUSv3]
cabsl(GLIBC_2.2) [SUSv3]	cacos(GLIBC_2.2) [SUSv3]	cacosf(GLIBC_2.2) [SUSv3]	cacosh(GLIBC_2.2) [SUSv3]
cacoshf(GLIBC_2.2) [SUSv3]	cacoshl(GLIBC_2.2) [SUSv3]	cacosl(GLIBC_2.2) [SUSv3]	carg(GLIBC_2.2) [SUSv3]
cargf(GLIBC_2.2) [SUSv3]	cargl(GLIBC_2.2) [SUSv3]	casin(GLIBC_2.2) [SUSv3]	casinf(GLIBC_2.2) [SUSv3]
casinh(GLIBC_2.2) [SUSv3]	casinhf(GLIBC_2.2) [SUSv3]	casinhL(GLIBC_2.2) [SUSv3]	casinl(GLIBC_2.2) [SUSv3]
catan(GLIBC_2.2) [SUSv3]	catanf(GLIBC_2.2) [SUSv3]	catanh(GLIBC_2.2) [SUSv3]	catanhf(GLIBC_2.2) [SUSv3]
catanhl(GLIBC_2.2) [SUSv3]	catanl(GLIBC_2.2) [SUSv3]	cbrt(GLIBC_2.2) [SUSv3]	cbrtf(GLIBC_2.2) [SUSv3]
cbrtl(GLIBC_2.2) [SUSv3]	ccos(GLIBC_2.2) [SUSv3]	ccosf(GLIBC_2.2) [SUSv3]	ccosh(GLIBC_2.2) [SUSv3]
ccoshf(GLIBC_2.2) [SUSv3]	ccoshl(GLIBC_2.2) [SUSv3]	ccosl(GLIBC_2.2) [SUSv3]	ceil(GLIBC_2.2) [SUSv3]
ceilf(GLIBC_2.2) [SUSv3]	ceill(GLIBC_2.2) [SUSv3]	cexp(GLIBC_2.2) [SUSv3]	cexpf(GLIBC_2.2) [SUSv3]
cexpl(GLIBC_2.2) [SUSv3]	cimag(GLIBC_2.2) [SUSv3]	cimagf(GLIBC_2.2) [SUSv3]	cimagl(GLIBC_2.2) [SUSv3]
clog(GLIBC_2.2) [SUSv3]	clog10(GLIBC_2.2) [ISOC99]	clog10f(GLIBC_2.2) [ISOC99]	clog10l(GLIBC_2.2) [ISOC99]
clogf(GLIBC_2.2) [SUSv3]	clogl(GLIBC_2.2) [SUSv3]	conj(GLIBC_2.2) [SUSv3]	conjf(GLIBC_2.2) [SUSv3]
conjl(GLIBC_2.2) [SUSv3]	copysign(GLIBC_2.2) [SUSv3]	copysignf(GLIBC_2.2) [SUSv3]	copysignl(GLIBC_2.2) [SUSv3]
cos(GLIBC_2.2) [SUSv3]	cosf(GLIBC_2.2) [SUSv3]	cosh(GLIBC_2.2) [SUSv3]	coshf(GLIBC_2.2) [SUSv3]

coshl(GLIBC_2.2) [SUSv3]	cosl(GLIBC_2.2) [SUSv3]	cpow(GLIBC_2.2) [SUSv3]	cpowf(GLIBC_2.2) [SUSv3]
cpowl(GLIBC_2.2) [SUSv3]	cproj(GLIBC_2.2) [SUSv3]	cprojf(GLIBC_2.2) [SUSv3]	cprojl(GLIBC_2.2) [SUSv3]
creal(GLIBC_2.2) [SUSv3]	crealf(GLIBC_2.2) [SUSv3]	creall(GLIBC_2.2) [SUSv3]	csin(GLIBC_2.2) [SUSv3]
csinf(GLIBC_2.2) [SUSv3]	csinh(GLIBC_2.2) [SUSv3]	csinhf(GLIBC_2.2) [SUSv3]	csinhl(GLIBC_2.2) [SUSv3]
csinl(GLIBC_2.2) [SUSv3]	csqrt(GLIBC_2.2) [SUSv3]	csqrtf(GLIBC_2.2) [SUSv3]	csqrtl(GLIBC_2.2) [SUSv3]
ctan(GLIBC_2.2) [SUSv3]	ctanf(GLIBC_2.2) [SUSv3]	ctanh(GLIBC_2.2) [SUSv3]	ctanhf(GLIBC_2.2) [SUSv3]
ctanhl(GLIBC_2.2) [SUSv3]	ctanl(GLIBC_2.2) [SUSv3]	dremf(GLIBC_2.2) [ISOC99]	dreml(GLIBC_2.2) [ISOC99]
erf(GLIBC_2.2) [SUSv3]	erfc(GLIBC_2.2) [SUSv3]	erfcf(GLIBC_2.2) [SUSv3]	erfcl(GLIBC_2.2) [SUSv3]
erff(GLIBC_2.2) [SUSv3]	erfl(GLIBC_2.2) [SUSv3]	exp(GLIBC_2.2) [SUSv3]	exp2(GLIBC_2.2) [SUSv3]
exp2f(GLIBC_2.2) [SUSv3]	exp2l(GLIBC_2.2) [SUSv3]	expf(GLIBC_2.2) [SUSv3]	expl(GLIBC_2.2) [SUSv3]
expm1(GLIBC_2.2) [SUSv3]	expm1f(GLIBC_2.2) [SUSv3]	expm1l(GLIBC_2.2) [SUSv3]	fabs(GLIBC_2.2) [SUSv3]
fabsf(GLIBC_2.2) [SUSv3]	fabsl(GLIBC_2.2) [SUSv3]	fdim(GLIBC_2.2) [SUSv3]	fdimf(GLIBC_2.2) [SUSv3]
fdiml(GLIBC_2.2) [SUSv3]	feclearexcept(GLIBC_2.2) [SUSv3]	fegetenv(GLIBC_2.2) [SUSv3]	fegetexceptflag(GLIBC_2.2) [SUSv3]
fegetround(GLIBC_2.2) [SUSv3]	fehldexcept(GLIBC_2.2) [SUSv3]	feraiseexcept(GLIBC_2.2) [SUSv3]	fesetenv(GLIBC_2.2) [SUSv3]
fesetexceptflag(GLIBC_2.2) [SUSv3]	fesetround(GLIBC_2.2) [SUSv3]	fetestexcept(GLIBC_2.2) [SUSv3]	feupdateenv(GLIBC_2.2) [SUSv3]
finite(GLIBC_2.2) [SUSv2]	finitef(GLIBC_2.2) [ISOC99]	finitel(GLIBC_2.2) [ISOC99]	floor(GLIBC_2.2) [SUSv3]
floorf(GLIBC_2.2) [SUSv3]	floorl(GLIBC_2.2) [SUSv3]	fma(GLIBC_2.2) [SUSv3]	fmaf(GLIBC_2.2) [SUSv3]
fmal(GLIBC_2.2) [SUSv3]	fmax(GLIBC_2.2) [SUSv3]	fmaxf(GLIBC_2.2) [SUSv3]	fmaxl(GLIBC_2.2) [SUSv3]
fmin(GLIBC_2.2) [SUSv3]	fminf(GLIBC_2.2) [SUSv3]	fminl(GLIBC_2.2) [SUSv3]	fmod(GLIBC_2.2) [SUSv3]
fmodf(GLIBC_2.2) [SUSv3]	fmodl(GLIBC_2.2) [SUSv3]	frexp(GLIBC_2.2) [SUSv3]	frexpf(GLIBC_2.2) [SUSv3]
frexpl(GLIBC_2.2)	gamma(GLIBC_2.2)	gammaf(GLIBC_2.2)	gammal(GLIBC_2.2)

[SUSv3]	2) [SUSv2]	.2) [ISOC99]	.2) [ISOC99]
hypot(GLIBC_2.2) [SUSv3]	hypotf(GLIBC_2.2) [SUSv3]	hypotl(GLIBC_2.2) [SUSv3]	ilogb(GLIBC_2.2) [SUSv3]
ilogbf(GLIBC_2.2) [SUSv3]	ilogbl(GLIBC_2.2) [SUSv3]	j0(GLIBC_2.2) [SUSv3]	j0f(GLIBC_2.2) [ISOC99]
j0l(GLIBC_2.2) [ISOC99]	j1(GLIBC_2.2) [SUSv3]	j1f(GLIBC_2.2) [ISOC99]	j1l(GLIBC_2.2) [ISOC99]
jn(GLIBC_2.2) [SUSv3]	jnf(GLIBC_2.2) [ISOC99]	jnl(GLIBC_2.2) [ISOC99]	ldexp(GLIBC_2.2) [SUSv3]
ldexpf(GLIBC_2.2) [SUSv3]	ldexpl(GLIBC_2.2) [SUSv3]	lgamma(GLIBC_2.2) [SUSv3]	lgamma_r(GLIBC_2.2) [ISOC99]
lgammaf(GLIBC_2.2) [SUSv3]	lgammaf_r(GLIBC_2.2) [ISOC99]	lgammal(GLIBC_2.2) [SUSv3]	lgammal_r(GLIBC_2.2) [ISOC99]
llrint(GLIBC_2.2) [SUSv3]	llrintf(GLIBC_2.2) [SUSv3]	llrintl(GLIBC_2.2) [SUSv3]	llround(GLIBC_2.2) [SUSv3]
llroundf(GLIBC_2.2) [SUSv3]	llroundl(GLIBC_2.2) [SUSv3]	log(GLIBC_2.2) [SUSv3]	log10(GLIBC_2.2) [SUSv3]
log10f(GLIBC_2.2) [SUSv3]	log10l(GLIBC_2.2) [SUSv3]	log1p(GLIBC_2.2) [SUSv3]	log1pf(GLIBC_2.2) [SUSv3]
log1pl(GLIBC_2.2) [SUSv3]	log2(GLIBC_2.2) [SUSv3]	log2f(GLIBC_2.2) [SUSv3]	log2l(GLIBC_2.2) [SUSv3]
logb(GLIBC_2.2) [SUSv3]	logbf(GLIBC_2.2) [SUSv3]	logbl(GLIBC_2.2) [SUSv3]	logf(GLIBC_2.2) [SUSv3]
logl(GLIBC_2.2) [SUSv3]	lrint(GLIBC_2.2) [SUSv3]	lrintf(GLIBC_2.2) [SUSv3]	lrintl(GLIBC_2.2) [SUSv3]
lround(GLIBC_2.2) [SUSv3]	lroundf(GLIBC_2.2) [SUSv3]	lroundl(GLIBC_2.2) [SUSv3]	matherr(GLIBC_2.2) [ISOC99]
modf(GLIBC_2.2) [SUSv3]	modff(GLIBC_2.2) [SUSv3]	modfl(GLIBC_2.2) [SUSv3]	nan(GLIBC_2.2) [SUSv3]
nanf(GLIBC_2.2) [SUSv3]	nanl(GLIBC_2.2) [SUSv3]	nearbyint(GLIBC_2.2) [SUSv3]	nearbyintf(GLIBC_2.2) [SUSv3]
nearbyintl(GLIBC_2.2) [SUSv3]	nextafter(GLIBC_2.2) [SUSv3]	nextafterf(GLIBC_2.2) [SUSv3]	nextafterl(GLIBC_2.2) [SUSv3]
nexttoward(GLIBC_2.2) [SUSv3]	nexttowardf(GLIBC_2.2) [SUSv3]	nexttowardl(GLIBC_2.2) [SUSv3]	pow(GLIBC_2.2) [SUSv3]
pow10(GLIBC_2.2) [ISOC99]	pow10f(GLIBC_2.2) [ISOC99]	pow10l(GLIBC_2.2) [ISOC99]	powf(GLIBC_2.2) [SUSv3]
powl(GLIBC_2.2) [SUSv3]	remainder(GLIBC_2.2) [SUSv3]	remainderf(GLIBC_2.2) [SUSv3]	remainderl(GLIBC_2.2) [SUSv3]
remquo(GLIBC_2.2) [SUSv3]	remquof(GLIBC_2.2) [SUSv3]	remquol(GLIBC_2.2) [SUSv3]	rint(GLIBC_2.2) [SUSv3]

rintf(GLIBC_2.2) [SUSv3]	rintl(GLIBC_2.2) [SUSv3]	round(GLIBC_2.2) [SUSv3]	roundf(GLIBC_2.2) [SUSv3]
roundl(GLIBC_2.2) [SUSv3]	scalb(GLIBC_2.2) [SUSv3]	scalbf(GLIBC_2.2) [ISOC99]	scalbl(GLIBC_2.2) [ISOC99]
scalbln(GLIBC_2.2) [SUSv3]	scalblnf(GLIBC_2.2) [SUSv3]	scalblnl(GLIBC_2.2) [SUSv3]	scalbn(GLIBC_2.2) [SUSv3]
scalbnf(GLIBC_2.2) [SUSv3]	scalbnl(GLIBC_2.2) [SUSv3]	significantd(GLIBC_2.2) [ISOC99]	significantdf(GLIBC_2.2) [ISOC99]
significantdl(GLIBC_2.2) [ISOC99]	sin(GLIBC_2.2) [SUSv3]	sincos(GLIBC_2.2) [ISOC99]	sincosf(GLIBC_2.2) [ISOC99]
sincosl(GLIBC_2.2) [ISOC99]	sinf(GLIBC_2.2) [SUSv3]	sinh(GLIBC_2.2) [SUSv3]	sinhf(GLIBC_2.2) [SUSv3]
sinhl(GLIBC_2.2) [SUSv3]	sinl(GLIBC_2.2) [SUSv3]	sqrt(GLIBC_2.2) [SUSv3]	sqrtf(GLIBC_2.2) [SUSv3]
sqrtl(GLIBC_2.2) [SUSv3]	tan(GLIBC_2.2) [SUSv3]	tanf(GLIBC_2.2) [SUSv3]	tanh(GLIBC_2.2) [SUSv3]
tanhf(GLIBC_2.2) [SUSv3]	tanhL(GLIBC_2.2) [SUSv3]	tanl(GLIBC_2.2) [SUSv3]	tgamma(GLIBC_2.2) [SUSv3]
tgammaf(GLIBC_2.2) [SUSv3]	tgammal(GLIBC_2.2) [SUSv3]	trunc(GLIBC_2.2) [SUSv3]	truncf(GLIBC_2.2) [SUSv3]
truncl(GLIBC_2.2) [SUSv3]	y0(GLIBC_2.2) [SUSv3]	y0f(GLIBC_2.2) [ISOC99]	y0l(GLIBC_2.2) [ISOC99]
y1(GLIBC_2.2) [SUSv3]	y1f(GLIBC_2.2) [ISOC99]	y1l(GLIBC_2.2) [ISOC99]	yn(GLIBC_2.2) [SUSv3]
ynf(GLIBC_2.2) [ISOC99]	ynl(GLIBC_2.2) [ISOC99]		

1447

1448

1449

1450

An LSB conforming implementation shall provide the architecture specific data interfaces for Math specified in Table 11-26, with the full mandatory functionality as described in the referenced underlying specification.

1451

**Table 11-26 libm - Math Data Interfaces**

signgam(GLIBC_2.2) [SUSv3]			
----------------------------	--	--	--

1452

### 11.5 Data Definitions for libm

1453

1454

1455

1456

1457

1458

This section defines global identifiers and their values that are associated with interfaces contained in libm. These definitions are organized into groups that correspond to system headers. This convention is used as a convenience for the reader, and does not imply the existence of these headers, or their content. Where an interface is defined as requiring a particular system header file all of the data definitions for that system header file presented here shall be in effect.

1459

1460

This section gives data definitions to promote binary application portability, not to repeat source interface definitions available elsewhere. System providers and

1461 application developers should use this ABI to supplement - not to replace - source  
1462 interface definition specifications.

1463 This specification uses the ISO C (1999) C Language as the reference programming  
1464 language, and data definitions are specified in ISO C format. The C language is used  
1465 here as a convenient notation. Using a C language description of these data objects  
1466 does not preclude their use by other programming languages.

### 11.5.1 complex.h

```

1467
1468 extern double cabs(double complex);
1469 extern float cabsf(float complex);
1470 extern long double cabsl(long double complex);
1471 extern double complex cacos(double complex);
1472 extern float complex cacosf(float complex);
1473 extern double complex cacosh(double complex);
1474 extern float complex cacoshf(float complex);
1475 extern long double complex cacoshl(long double complex);
1476 extern long double complex cacosl(long double complex);
1477 extern double carg(double complex);
1478 extern float cargf(float complex);
1479 extern long double cargl(long double complex);
1480 extern double complex casin(double complex);
1481 extern float complex casinf(float complex);
1482 extern double complex casinh(double complex);
1483 extern float complex casinhf(float complex);
1484 extern long double complex casinhl(long double complex);
1485 extern long double complex casinl(long double complex);
1486 extern double complex catan(double complex);
1487 extern float complex catanf(float complex);
1488 extern double complex catanh(double complex);
1489 extern float complex catanhf(float complex);
1490 extern long double complex catanhl(long double complex);
1491 extern long double complex catanl(long double complex);
1492 extern double complex ccos(double complex);
1493 extern float complex ccosf(float complex);
1494 extern double complex ccosh(double complex);
1495 extern float complex ccoshf(float complex);
1496 extern long double complex ccoshl(long double complex);
1497 extern long double complex ccosl(long double complex);
1498 extern double complex cexp(double complex);
1499 extern float complex cexpf(float complex);
1500 extern long double complex cexpl(long double complex);
1501 extern double cimag(double complex);
1502 extern float cimagf(float complex);
1503 extern long double cimagl(long double complex);
1504 extern double complex clog(double complex);
1505 extern float complex clog10f(float complex);
1506 extern long double complex clog10l(long double complex);
1507 extern float complex clogf(float complex);
1508 extern long double complex clogl(long double complex);
1509 extern double complex conj(double complex);
1510 extern float complex conjf(float complex);
1511 extern long double complex conjl(long double complex);
1512 extern double complex cpow(double complex, double complex);
1513 extern float complex cpowf(float complex, float complex);
1514 extern long double complex cpowl(long double complex, long double
1515 complex);
1516 extern double complex cproj(double complex);
1517 extern float complex cprojf(float complex);
1518 extern long double complex cprojl(long double complex);
1519 extern double creal(double complex);

```

```

1520     extern float crealf(float complex);
1521     extern long double creall(long double complex);
1522     extern double complex csin(double complex);
1523     extern float complex csinf(float complex);
1524     extern double complex csinh(double complex);
1525     extern float complex csinhf(float complex);
1526     extern long double complex csinhl(long double complex);
1527     extern long double complex csinl(long double complex);
1528     extern double complex csqrt(double complex);
1529     extern float complex csqrtf(float complex);
1530     extern long double complex csqrtl(long double complex);
1531     extern double complex ctan(double complex);
1532     extern float complex ctanf(float complex);
1533     extern double complex ctanh(double complex);
1534     extern float complex ctanhf(float complex);
1535     extern long double complex ctanhl(long double complex);
1536     extern long double complex ctanl(long double complex);

```

### 11.5.2 fenv.h

```

1537
1538     #define FE_INVALID      (1UL << 0)
1539     #define FE_DIVBYZERO   (1UL << 2)
1540     #define FE_OVERFLOW    (1UL << 3)
1541     #define FE_UNDERFLOW  (1UL << 4)
1542     #define FE_INEXACT    (1UL << 5)
1543     #define FE_UNNORMAL    1UL << 1
1544
1545     #define FE_ALL_EXCEPT \
1546         (FE_INEXACT | FE_UNDERFLOW | FE_OVERFLOW | FE_DIVBYZERO |
1547         FE_UNNORMAL | FE_INVALID)
1548
1549     #define FE_TONEAREST   0
1550     #define FE_DOWNWARD   1
1551     #define FE_UPWARD     2
1552     #define FE_TOWARDZERO 3
1553
1554     typedef unsigned long int fexcept_t;
1555
1556     typedef unsigned long int fenv_t;
1557
1558     #define FE_DFL_ENV      ((__const fenv_t *) 0xc009804c0270033FUL)
1559
1560     extern int feclearexcept(int);
1561     extern int fegetenv(fenv_t *);
1562     extern int fegetexceptflag(fexcept_t *, int);
1563     extern int fegetround(void);
1564     extern int feholdexcept(fenv_t *);
1565     extern int feraiseexcept(int);
1566     extern int fesetenv(const fenv_t *);
1567     extern int fesetexceptflag(const fexcept_t *, int);
1568     extern int fesetround(int);
1569     extern int fetestexcept(int);
1570     extern int feupdateenv(const fenv_t *);

```

### 11.5.3 math.h

```

1571
1572     #define fpclassify(x) \
1573         (sizeof (x) == sizeof (float) ? __fpclassifyf (x) : sizeof (x)
1574         == sizeof (double) ? __fpclassify (x) : __fpclassifyl (x))
1575     #define signbit(x) \
1576         (sizeof (x) == sizeof (float)? __signbitf (x) : sizeof (x) ==
1577         sizeof (double)? __signbit (x) : __signbitl (x))

```

```

1578
1579     #define FP_ILOGB0      -2147483648
1580     #define FP_ILOGBNAN   2147483647
1581
1582     extern int __finite(double);
1583     extern int __finitef(float);
1584     extern int __finitel(long double);
1585     extern int __isinf(double);
1586     extern int __isinff(float);
1587     extern int __isinfl(long double);
1588     extern int __isnan(double);
1589     extern int __isnanf(float);
1590     extern int __isnanl(long double);
1591     extern int __signbit(double);
1592     extern int __signbitf(float);
1593     extern int __fpclassify(double);
1594     extern int __fpclassifyf(float);
1595     extern int __fpclassifyl(long double);
1596     extern int signgam(void);
1597     extern double copysign(double, double);
1598     extern int finite(double);
1599     extern double frexp(double, int *);
1600     extern double ldexp(double, int);
1601     extern double modf(double, double *);
1602     extern double acos(double);
1603     extern double acosh(double);
1604     extern double asinh(double);
1605     extern double atanh(double);
1606     extern double asin(double);
1607     extern double atan(double);
1608     extern double atan2(double, double);
1609     extern double cbrt(double);
1610     extern double ceil(double);
1611     extern double cos(double);
1612     extern double cosh(double);
1613     extern double erf(double);
1614     extern double erfc(double);
1615     extern double exp(double);
1616     extern double expm1(double);
1617     extern double fabs(double);
1618     extern double floor(double);
1619     extern double fmod(double, double);
1620     extern double gamma(double);
1621     extern double hypot(double, double);
1622     extern int ilogb(double);
1623     extern double j0(double);
1624     extern double j1(double);
1625     extern double jn(int, double);
1626     extern double lgamma(double);
1627     extern double log(double);
1628     extern double log10(double);
1629     extern double loglp(double);
1630     extern double logb(double);
1631     extern double nextafter(double, double);
1632     extern double pow(double, double);
1633     extern double remainder(double, double);
1634     extern double rint(double);
1635     extern double scalb(double, double);
1636     extern double sin(double);
1637     extern double sinh(double);
1638     extern double sqrt(double);
1639     extern double tan(double);
1640     extern double tanh(double);
1641     extern double y0(double);

```

## 11 Libraries

```
1642     extern double yl(double);
1643     extern double yn(int, double);
1644     extern float copysignf(float, float);
1645     extern long double copysignl(long double, long double);
1646     extern int finitef(float);
1647     extern int finitel(long double);
1648     extern float frexpf(float, int *);
1649     extern long double frexpl(long double, int *);
1650     extern float ldexpf(float, int);
1651     extern long double ldexpl(long double, int);
1652     extern float modff(float, float *);
1653     extern long double modfl(long double, long double *);
1654     extern double scalbln(double, long int);
1655     extern float scalblnf(float, long int);
1656     extern long double scalblnl(long double, long int);
1657     extern double scalbn(double, int);
1658     extern float scalbnf(float, int);
1659     extern long double scalbnl(long double, int);
1660     extern float acosf(float);
1661     extern float acoshf(float);
1662     extern long double acoshl(long double);
1663     extern long double acosl(long double);
1664     extern float asinf(float);
1665     extern float asinhf(float);
1666     extern long double asinhl(long double);
1667     extern long double asinl(long double);
1668     extern float atan2f(float, float);
1669     extern long double atan2l(long double, long double);
1670     extern float atanf(float);
1671     extern float atanhf(float);
1672     extern long double atanhhl(long double);
1673     extern long double atanl(long double);
1674     extern float cbrtf(float);
1675     extern long double cbrtl(long double);
1676     extern float ceilf(float);
1677     extern long double ceill(long double);
1678     extern float cosf(float);
1679     extern float coshf(float);
1680     extern long double coshl(long double);
1681     extern long double cosl(long double);
1682     extern float dremf(float, float);
1683     extern long double dreml(long double, long double);
1684     extern float erfcf(float);
1685     extern long double erfcl(long double);
1686     extern float erff(float);
1687     extern long double erfl(long double);
1688     extern double exp2(double);
1689     extern float exp2f(float);
1690     extern long double exp2l(long double);
1691     extern float expf(float);
1692     extern long double expl(long double);
1693     extern float expmlf(float);
1694     extern long double expmll(long double);
1695     extern float fabsf(float);
1696     extern long double fabsl(long double);
1697     extern double fdim(double, double);
1698     extern float fdimf(float, float);
1699     extern long double fdiml(long double, long double);
1700     extern float floorf(float);
1701     extern long double floorl(long double);
1702     extern double fma(double, double, double);
1703     extern float fmaf(float, float, float);
1704     extern long double fmal(long double, long double, long double);
1705     extern double fmax(double, double);
```



```

1706     extern float fmaxf(float, float);
1707     extern long double fmaxl(long double, long double);
1708     extern double fmin(double, double);
1709     extern float fminf(float, float);
1710     extern long double fminl(long double, long double);
1711     extern float fmodf(float, float);
1712     extern long double fmodl(long double, long double);
1713     extern float gammaf(float);
1714     extern long double gammal(long double);
1715     extern float hypotf(float, float);
1716     extern long double hypotl(long double, long double);
1717     extern int ilogbf(float);
1718     extern int ilogbl(long double);
1719     extern float j0f(float);
1720     extern long double j0l(long double);
1721     extern float j1f(float);
1722     extern long double j1l(long double);
1723     extern float jnf(int, float);
1724     extern long double jnl(int, long double);
1725     extern double lgamma_r(double, int *);
1726     extern float lgammaf(float);
1727     extern float lgammaf_r(float, int *);
1728     extern long double lgammal(long double);
1729     extern long double lgammal_r(long double, int *);
1730     extern long int llrint(double);
1731     extern long long int llrintf(float);
1732     extern long long int llrintl(long double);
1733     extern long long int llround(double);
1734     extern long long int llroundf(float);
1735     extern long long int llroundl(long double);
1736     extern float log10f(float);
1737     extern long double log10l(long double);
1738     extern float log1pf(float);
1739     extern long double log1pl(long double);
1740     extern double log2(double);
1741     extern float log2f(float);
1742     extern long double log2l(long double);
1743     extern float logbf(float);
1744     extern long double logbl(long double);
1745     extern float logf(float);
1746     extern long double logl(long double);
1747     extern long int lrint(double);
1748     extern long int lrintf(float);
1749     extern long int lrintl(long double);
1750     extern long int lround(double);
1751     extern long int lroundf(float);
1752     extern long int lroundl(long double);
1753     extern int matherr(struct exception *);
1754     extern double nan(const char *);
1755     extern float nanf(const char *);
1756     extern long double nanl(const char *);
1757     extern double nearbyint(double);
1758     extern float nearbyintf(float);
1759     extern long double nearbyintl(long double);
1760     extern float nextafterf(float, float);
1761     extern long double nextafterl(long double, long double);
1762     extern double nexttoward(double, long double);
1763     extern float nexttowardf(float, long double);
1764     extern long double nexttowardl(long double, long double);
1765     extern double pow10(double);
1766     extern float pow10f(float);
1767     extern long double pow10l(long double);
1768     extern float powf(float, float);
1769     extern long double powl(long double, long double);

```

```

1770     extern float remainderf(float, float);
1771     extern long double remainderl(long double, long double);
1772     extern double remquo(double, double, int *);
1773     extern float remquof(float, float, int *);
1774     extern long double remquo1(long double, long double, int *);
1775     extern float rintf(float);
1776     extern long double rintl(long double);
1777     extern double round(double);
1778     extern float roundf(float);
1779     extern long double roundl(long double);
1780     extern float scalbf(float, float);
1781     extern long double scalbl(long double, long double);
1782     extern double significand(double);
1783     extern float significandf(float);
1784     extern long double significandl(long double);
1785     extern void sincos(double, double *, double *);
1786     extern void sincosf(float, float *, float *);
1787     extern void sincosl(long double, long double *, long double *);
1788     extern float sinf(float);
1789     extern float sinhf(float);
1790     extern long double sinhl(long double);
1791     extern long double sinl(long double);
1792     extern float sqrtf(float);
1793     extern long double sqrtl(long double);
1794     extern float tanf(float);
1795     extern float tanhf(float);
1796     extern long double tanhl(long double);
1797     extern long double tanl(long double);
1798     extern double tgamma(double);
1799     extern float tgammaf(float);
1800     extern long double tgammal(long double);
1801     extern double trunc(double);
1802     extern float truncf(float);
1803     extern long double trunc1(long double);
1804     extern float y0f(float);
1805     extern long double y0l(long double);
1806     extern float y1f(float);
1807     extern long double y1l(long double);
1808     extern float ynf(int, float);
1809     extern long double ynl(int, long double);
1810     extern int __fpclassifyl(long double);
1811     extern int __fpclassifyl(long double);
1812     extern int __signbitl(long double);
1813     extern int __signbitl(long double);
1814     extern int __signbitl(long double);
1815     extern long double exp2l(long double);
1816     extern long double exp2l(long double);

```

## 11.6 Interfaces for libpthread

1817 Table 11-27 defines the library name and shared object name for the libpthread  
1818 library

1819 **Table 11-27 libpthread Definition**

Library:	libpthread
SONAME:	libpthread.so.0

1821 The behavior of the interfaces in this library is specified by the following specifica-  
1822 tions:

[LFS] Large File Support

[LSB] This Specification  
[SUSv3] ISO POSIX (2003)

1823

## 11.6.1 Realtime Threads

1824

### 11.6.1.1 Interfaces for Realtime Threads

1825

An LSB conforming implementation shall provide the architecture specific functions for Realtime Threads specified in Table 11-28, with the full mandatory functionality as described in the referenced underlying specification.

1826

1827

1828

**Table 11-28 libpthread - Realtime Threads Function Interfaces**

pthread_attr_getinheritsched(GLIBC_2.2) [SUSv3]	pthread_attr_getschedpolicy(GLIBC_2.2) [SUSv3]	pthread_attr_getscope(GLIBC_2.2) [SUSv3]	pthread_attr_setinheritsched(GLIBC_2.2) [SUSv3]
pthread_attr_setschedpolicy(GLIBC_2.2) [SUSv3]	pthread_attr_setscope(GLIBC_2.2) [SUSv3]	pthread_getschedparam(GLIBC_2.2) [SUSv3]	pthread_setschedparam(GLIBC_2.2) [SUSv3]

1829

## 11.6.2 Advanced Realtime Threads

1830

### 11.6.2.1 Interfaces for Advanced Realtime Threads

1831

No external functions are defined for libpthread - Advanced Realtime Threads in this part of the specification. See also the generic specification.

1832

## 11.6.3 Posix Threads

1833

### 11.6.3.1 Interfaces for Posix Threads

1834

An LSB conforming implementation shall provide the architecture specific functions for Posix Threads specified in Table 11-29, with the full mandatory functionality as described in the referenced underlying specification.

1835

1836

1837

**Table 11-29 libpthread - Posix Threads Function Interfaces**

_pthread_cleanup_pop(GLIBC_2.2) [LSB]	_pthread_cleanup_push(GLIBC_2.2) [LSB]	pthread_attr_destroy(GLIBC_2.2) [SUSv3]	pthread_attr_getdetachstate(GLIBC_2.2) [SUSv3]
pthread_attr_getguardsize(GLIBC_2.2) [SUSv3]	pthread_attr_getschedparam(GLIBC_2.2) [SUSv3]	pthread_attr_getstack(GLIBC_2.2) [SUSv3]	pthread_attr_getstackaddr(GLIBC_2.2) [SUSv3]
pthread_attr_getstacksize(GLIBC_2.2) [SUSv3]	pthread_attr_init(GLIBC_2.2) [SUSv3]	pthread_attr_setdetachstate(GLIBC_2.2) [SUSv3]	pthread_attr_setguardsize(GLIBC_2.2) [SUSv3]
pthread_attr_setschedparam(GLIBC_2.2) [SUSv3]	pthread_attr_setstackaddr(GLIBC_2.2) [SUSv3]	pthread_attr_setstacksize(GLIBC_2.3.3) [SUSv3]	pthread_cancel(GLIBC_2.2) [SUSv3]
pthread_cond_broadcast(GLIBC_2.3.2) [SUSv3]	pthread_cond_destroy(GLIBC_2.3.2) [SUSv3]	pthread_cond_init(GLIBC_2.3.2) [SUSv3]	pthread_cond_signal(GLIBC_2.3.2) [SUSv3]

pthread_cond_timedwait(GLIBC_2.3.2) [SUSv3]	pthread_cond_wait(GLIBC_2.3.2) [SUSv3]	pthread_condattr_destroy(GLIBC_2.2) [SUSv3]	pthread_condattr_getpshared(GLIBC_2.2) [SUSv3]
pthread_condattr_init(GLIBC_2.2) [SUSv3]	pthread_condattr_setpshared(GLIBC_2.2) [SUSv3]	pthread_create(GLIBC_2.2) [SUSv3]	pthread_detach(GLIBC_2.2) [SUSv3]
pthread_equal(GLIBC_2.2) [SUSv3]	pthread_exit(GLIBC_2.2) [SUSv3]	pthread_getconcurrency(GLIBC_2.2) [SUSv3]	pthread_getspecific(GLIBC_2.2) [SUSv3]
pthread_join(GLIBC_2.2) [SUSv3]	pthread_key_create(GLIBC_2.2) [SUSv3]	pthread_key_delete(GLIBC_2.2) [SUSv3]	pthread_kill(GLIBC_2.2) [SUSv3]
pthread_mutex_destroy(GLIBC_2.2) [SUSv3]	pthread_mutex_init(GLIBC_2.2) [SUSv3]	pthread_mutex_lock(GLIBC_2.2) [SUSv3]	pthread_mutex_trylock(GLIBC_2.2) [SUSv3]
pthread_mutex_unlock(GLIBC_2.2) [SUSv3]	pthread_mutexattr_destroy(GLIBC_2.2) [SUSv3]	pthread_mutexattr_getpshared(GLIBC_2.2) [SUSv3]	pthread_mutexattr_gettype(GLIBC_2.2) [SUSv3]
pthread_mutexattr_init(GLIBC_2.2) [SUSv3]	pthread_mutexattr_setpshared(GLIBC_2.2) [SUSv3]	pthread_mutexattr_settype(GLIBC_2.2) [SUSv3]	pthread_once(GLIBC_2.2) [SUSv3]
pthread_rwlock_destroy(GLIBC_2.2) [SUSv3]	pthread_rwlock_init(GLIBC_2.2) [SUSv3]	pthread_rwlock_rdlock(GLIBC_2.2) [SUSv3]	pthread_rwlock_timedrdlock(GLIBC_2.2) [SUSv3]
pthread_rwlock_timedwrlock(GLIBC_2.2) [SUSv3]	pthread_rwlock_tryrdlock(GLIBC_2.2) [SUSv3]	pthread_rwlock_trywrlock(GLIBC_2.2) [SUSv3]	pthread_rwlock_unlock(GLIBC_2.2) [SUSv3]
pthread_rwlock_wrlock(GLIBC_2.2) [SUSv3]	pthread_rwlockattr_destroy(GLIBC_2.2) [SUSv3]	pthread_rwlockattr_getpshared(GLIBC_2.2) [SUSv3]	pthread_rwlockattr_init(GLIBC_2.2) [SUSv3]
pthread_rwlockattr_setpshared(GLIBC_2.2) [SUSv3]	pthread_self(GLIBC_2.2) [SUSv3]	pthread_setcancelstate(GLIBC_2.2) [SUSv3]	pthread_setcanceltype(GLIBC_2.2) [SUSv3]
pthread_setconcurrency(GLIBC_2.2) [SUSv3]	pthread_setspecific(GLIBC_2.2) [SUSv3]	pthread_sigmask(GLIBC_2.2) [SUSv3]	pthread_testcancel(GLIBC_2.2) [SUSv3]
sem_close(GLIBC_2.2) [SUSv3]	sem_destroy(GLIBC_2.2) [SUSv3]	sem_getvalue(GLIBC_2.2) [SUSv3]	sem_init(GLIBC_2.2) [SUSv3]
sem_open(GLIBC_2.2) [SUSv3]	sem_post(GLIBC_2.2) [SUSv3]	sem_timedwait(GLIBC_2.2) [SUSv3]	sem_trywait(GLIBC_2.2) [SUSv3]
sem_unlink(GLIBC_2.2) [SUSv3]	sem_wait(GLIBC_2.2) [SUSv3]		

## 11.6.4 Thread aware versions of libc interfaces

1839

### 11.6.4.1 Interfaces for Thread aware versions of libc interfaces

1840

An LSB conforming implementation shall provide the architecture specific functions for Thread aware versions of libc interfaces specified in Table 11-30, with the full mandatory functionality as described in the referenced underlying specification.

1841

1842

1843

**Table 11-30 libpthread - Thread aware versions of libc interfaces Function Interfaces**

1844

lseek64(GLIBC_2.2) [LFS]	open64(GLIBC_2.2) [LFS]	pread(GLIBC_2.2) [SUSv3]	pread64(GLIBC_2.2) [LFS]
pwrite(GLIBC_2.2) [SUSv3]	pwrite64(GLIBC_2.2) [LFS]		

1845

## 11.7 Data Definitions for libpthread

1846

This section defines global identifiers and their values that are associated with interfaces contained in libpthread. These definitions are organized into groups that correspond to system headers. This convention is used as a convenience for the reader, and does not imply the existence of these headers, or their content. Where an interface is defined as requiring a particular system header file all of the data definitions for that system header file presented here shall be in effect.

1847

1848

1849

1850

1851

1852

This section gives data definitions to promote binary application portability, not to repeat source interface definitions available elsewhere. System providers and application developers should use this ABI to supplement - not to replace - source interface definition specifications.

1853

1854

1855

1856

This specification uses the ISO C (1999) C Language as the reference programming language, and data definitions are specified in ISO C format. The C language is used here as a convenient notation. Using a C language description of these data objects does not preclude their use by other programming languages.

1857

1858

1859

### 11.7.1 pthread.h

1860

1861

```
extern void _pthread_cleanup_pop(struct _pthread_cleanup_buffer *,
int);
```

1862

1863

```
extern void _pthread_cleanup_push(struct _pthread_cleanup_buffer *,
void (*__routine) (void *)
, void *);
```

1864

1865

```
extern int pthread_attr_destroy(pthread_attr_t *);
```

1866

1867

```
extern int pthread_attr_getdetachstate(const typedef struct {
int __detachstate;
int __schedpolicy;
struct sched_param
```

1868

1869

```
__schedparam;
```

1870

1871

```
int __inheritsched;
int __scope;
size_t __guardsize;
int __stackaddr_set;
void *__stackaddr;
unsigned long int __stacksize;}
pthread_attr_t *, int *);
```

1872

1873

1874

1875

1876

1877

```
extern int pthread_attr_getinheritsched(const typedef struct {
int __detachstate;
int __schedpolicy;
```

1878

1879

1880

1881

```

1882                                     struct sched_param
1883     __schedparam;
1884                                     int __inheritsched;
1885                                     int __scope;
1886                                     size_t __guardsize;
1887                                     int __stackaddr_set;
1888                                     void *__stackaddr;
1889                                     unsigned long int
1890     __stacksize;}
1891                                     pthread_attr_t *, int *);
1892 extern int pthread_attr_getschedparam(const typedef struct {
1893     int __detachstate;
1894     int __schedpolicy;
1895     struct sched_param
1896     __schedparam;
1897                                     int __inheritsched;
1898                                     int __scope;
1899                                     size_t __guardsize;
1900                                     int __stackaddr_set;
1901                                     void *__stackaddr;
1902                                     unsigned long int __stacksize;}
1903     pthread_attr_t *, struct
1904     sched_param {
1905                                     int sched_priority;}
1906
1907                                     *);
1908 extern int pthread_attr_getschedpolicy(const typedef struct {
1909     int __detachstate;
1910     int __schedpolicy;
1911     struct sched_param
1912     __schedparam;
1913                                     int __inheritsched;
1914                                     int __scope;
1915                                     size_t __guardsize;
1916                                     int __stackaddr_set;
1917                                     void *__stackaddr;
1918                                     unsigned long int __stacksize;}
1919     pthread_attr_t *, int *);
1920 extern int pthread_attr_getscope(const typedef struct {
1921     int __detachstate;
1922     int __schedpolicy;
1923     struct sched_param __schedparam;
1924     int __inheritsched;
1925     int __scope;
1926     size_t __guardsize;
1927     int __stackaddr_set;
1928     void *__stackaddr;
1929     unsigned long int __stacksize;}
1930     pthread_attr_t *, int *);
1931 extern int pthread_attr_init(pthread_attr_t *);
1932 extern int pthread_attr_setdetachstate(pthread_attr_t *, int);
1933 extern int pthread_attr_setinheritsched(pthread_attr_t *, int);
1934 extern int pthread_attr_setschedparam(pthread_attr_t *, const struct
1935     sched_param {
1936                                     int sched_priority;}
1937
1938                                     *);
1939 extern int pthread_attr_setschedpolicy(pthread_attr_t *, int);
1940 extern int pthread_attr_setscope(pthread_attr_t *, int);
1941 extern int pthread_cancel(typedef unsigned long int pthread_t);
1942 extern int pthread_cond_broadcast(pthread_cond_t *);
1943 extern int pthread_cond_destroy(pthread_cond_t *);
1944 extern int pthread_cond_init(pthread_cond_t *, const typedef struct {
1945     int __dummy;}

```

```

1946
1947         pthread_condattr_t *);
1948 extern int pthread_cond_signal(pthread_cond_t *);
1949 extern int pthread_cond_timedwait(pthread_cond_t *, pthread_mutex_t *,
1950 const struct timespec {
1951         time_t tv_sec; long int tv_nsec;}
1952
1953         *);
1954 extern int pthread_cond_wait(pthread_cond_t *, pthread_mutex_t *);
1955 extern int pthread_condattr_destroy(pthread_condattr_t *);
1956 extern int pthread_condattr_init(pthread_condattr_t *);
1957 extern int pthread_create(pthread_t *, const typedef struct {
1958         int __detachstate;
1959         int __schedpolicy;
1960         struct sched_param __schedparam;
1961         int __inheritsched;
1962         int __scope;
1963         size_t __guardsize;
1964         int __stackaddr_set;
1965         void *__stackaddr;
1966         unsigned long int __stacksize;
1967         pthread_attr_t *,
1968         void *(*__start_routine) (void *p1)
1969         , void *);
1970 extern int pthread_detach(typedef unsigned long int pthread_t);
1971 extern int pthread_equal(typedef unsigned long int pthread_t,
1972         typedef unsigned long int pthread_t);
1973 extern void pthread_exit(void *);
1974 extern int pthread_getschedparam(typedef unsigned long int pthread_t,
1975         int *, struct sched_param {
1976         int sched_priority;}
1977
1978         *);
1979 extern void *pthread_getspecific(typedef unsigned int pthread_key_t);
1980 extern int pthread_join(typedef unsigned long int pthread_t, void **);
1981 extern int pthread_key_create(pthread_key_t *, void (*destr_func) (void
1982 *)
1983     );
1984 extern int pthread_key_delete(typedef unsigned int pthread_key_t);
1985 extern int pthread_mutex_destroy(pthread_mutex_t *);
1986 extern int pthread_mutex_init(pthread_mutex_t *, const typedef struct
1987 {
1988         int __mutexkind;}
1989
1990         pthread_mutexattr_t *);
1991 extern int pthread_mutex_lock(pthread_mutex_t *);
1992 extern int pthread_mutex_trylock(pthread_mutex_t *);
1993 extern int pthread_mutex_unlock(pthread_mutex_t *);
1994 extern int pthread_mutexattr_destroy(pthread_mutexattr_t *);
1995 extern int pthread_mutexattr_init(pthread_mutexattr_t *);
1996 extern int pthread_once(pthread_once_t *, void (*init_routine) (void)
1997     );
1998 extern int pthread_rwlock_destroy(pthread_rwlock_t *);
1999 extern int pthread_rwlock_init(pthread_rwlock_t *,
2000 pthread_rwlockattr_t *);
2001 extern int pthread_rwlock_rdlock(pthread_rwlock_t *);
2002 extern int pthread_rwlock_tryrdlock(pthread_rwlock_t *);
2003 extern int pthread_rwlock_trywrlock(pthread_rwlock_t *);
2004 extern int pthread_rwlock_unlock(pthread_rwlock_t *);
2005 extern int pthread_rwlock_wrlock(pthread_rwlock_t *);
2006 extern int pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
2007 extern int pthread_rwlockattr_getpshared(const typedef struct {
2008         int __lockkind; int
2009         __pshared;}

```

```

2010                                     pthread_rwlockattr_t *, int
2011 *);
2012 extern int pthread_rwlockattr_init(pthread_rwlockattr_t *);
2013 extern int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
2014 extern typedef unsigned long int pthread_t pthread_self(void);
2015 extern int pthread_setcancelstate(int, int *);
2016 extern int pthread_setcanceltype(int, int *);
2017 extern int pthread_setschedparam(pthread_t, int, const struct sched_param {
2018     int sched_priority;
2019     });
2020
2021 extern int pthread_setspecific(pthread_key_t, const void *);
2022
2023 extern void pthread_testcancel(void);
2024
2025 extern int pthread_attr_getguardsize(const pthread_attr_t {
2026     int __detachstate;
2027     int __schedpolicy;
2028     struct sched_param __schedparam;
2029     int __inheritsched;
2030     int __scope;
2031     size_t __guardsize;
2032     int __stackaddr_set;
2033     void *__stackaddr;
2034     unsigned long int __stacksize;
2035     pthread_attr_t *, size_t *);
2036 extern int pthread_attr_setguardsize(pthread_attr_t *,
2037     typedef unsigned long int
2038     size_t);
2039 extern int pthread_attr_setstackaddr(pthread_attr_t *, void *);
2040 extern int pthread_attr_getstackaddr(const pthread_attr_t {
2041     int __detachstate;
2042     int __schedpolicy;
2043     struct sched_param __schedparam;
2044     int __inheritsched;
2045     int __scope;
2046     size_t __guardsize;
2047     int __stackaddr_set;
2048     void *__stackaddr;
2049     unsigned long int __stacksize;
2050     pthread_attr_t *, void **);
2051 extern int pthread_attr_setstacksize(pthread_attr_t *,
2052     typedef unsigned long int
2053     size_t);
2054 extern int pthread_attr_getstacksize(const pthread_attr_t {
2055     int __detachstate;
2056     int __schedpolicy;
2057     struct sched_param __schedparam;
2058     int __inheritsched;
2059     int __scope;
2060     size_t __guardsize;
2061     int __stackaddr_set;
2062     void *__stackaddr;
2063     unsigned long int __stacksize;
2064     pthread_attr_t *, size_t *);
2065 extern int pthread_mutexattr_gettype(const pthread_mutexattr_t {
2066     int __mutexkind;
2067     pthread_mutexattr_t *, int *);
2068 extern int pthread_mutexattr_settype(pthread_mutexattr_t *, int);
2069 extern int pthread_getconcurrency(void);
2070 extern int pthread_setconcurrency(int);
2071 extern int pthread_attr_getstack(const pthread_attr_t {
2072     int __detachstate;
2073     int __schedpolicy;

```



```

2074         struct sched_param __schedparam;
2075         int __inheritsched;
2076         int __scope;
2077         size_t __guardsize;
2078         int __stackaddr_set;
2079         void *__stackaddr;
2080         unsigned long int __stacksize;}
2081         pthread_attr_t *, void **, size_t *);
2082 extern int pthread_attr_setstack(pthread_attr_t *, void *,
2083         typedef unsigned long int size_t);
2084 extern int pthread_condattr_getpshared(const typedef struct {
2085         int __dummy;}
2086         pthread_condattr_t *, int *);
2087 extern int pthread_condattr_setpshared(pthread_condattr_t *, int);
2088 extern int pthread_mutexattr_getpshared(const typedef struct {
2089         int __mutexkind;}
2090         pthread_mutexattr_t *, int *);
2091 extern int pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
2092 extern int pthread_rwlock_timedrdlock(pthread_rwlock_t *, const struct
2093 timespec {
2094         time_t tv_sec; long int
2095 tv_nsec;})
2096
2097         *);
2098 extern int pthread_rwlock_timedwrlock(pthread_rwlock_t *, const struct
2099 timespec {
2100         time_t tv_sec; long int
2101 tv_nsec;})
2102
2103         *);
2104 extern int __register_atfork(void (*prepare) (void)
2105         , void (*parent) (void)
2106         , void (*child) (void)
2107         , void *);
2108 extern int pthread_setschedprio(typedef unsigned long int pthread_t,
2109 int);

```

## 11.7.2 semaphore.h

```

2110
2111 extern int sem_close(sem_t *);
2112 extern int sem_destroy(sem_t *);
2113 extern int sem_getvalue(sem_t *, int *);
2114 extern int sem_init(sem_t *, int, unsigned int);
2115 extern sem_t *sem_open(const char *, int, ...);
2116 extern int sem_post(sem_t *);
2117 extern int sem_trywait(sem_t *);
2118 extern int sem_unlink(const char *);
2119 extern int sem_wait(sem_t *);
2120 extern int sem_timedwait(sem_t *, const struct timespec *);

```

## 11.8 Interfaces for libgcc\_s

2121 Table 11-31 defines the library name and shared object name for the libgcc\_s library

2122 **Table 11-31 libgcc\_s Definition**

Library:	libgcc_s
SONAME:	libgcc_s.so.1

2123

2124 The behavior of the interfaces in this library is specified by the following specifica-  
2125 tions:

2126 [LSB] This Specification

## 11.8.1 Unwind Library

### 11.8.1.1 Interfaces for Unwind Library

2128 An LSB conforming implementation shall provide the architecture specific functions  
2129 for Unwind Library specified in Table 11-32, with the full mandatory functionality as  
2130 described in the referenced underlying specification.

2131 **Table 11-32 libgcc\_s - Unwind Library Function Interfaces**

<code>_Unwind_Backtrace(GCC_3.3)</code> [LSB]	<code>_Unwind_DeleteException(GCC_3.0)</code> [LSB]	<code>_Unwind_FindEnclosingFunction(GCC_3.3)</code> [LSB]	<code>_Unwind_ForcedUnwind(GCC_3.0)</code> [LSB]
<code>_Unwind_GetBSP(GCC_3.3.2)</code> [LSB]	<code>_Unwind_GetCFA(GCC_3.3)</code> [LSB]	<code>_Unwind_GetGR(GCC_3.0)</code> [LSB]	<code>_Unwind_GetIP(GCC_3.0)</code> [LSB]
<code>_Unwind_GetLanguageSpecificData(GCC_3.0)</code> [LSB]	<code>_Unwind_GetRegionStart(GCC_3.0)</code> [LSB]	<code>_Unwind_RaiseException(GCC_3.0)</code> [LSB]	<code>_Unwind_Resume(GCC_3.0)</code> [LSB]
<code>_Unwind_Resume_or_Rethrow(GCC_3.3)</code> [LSB]	<code>_Unwind_SetGR(GCC_3.0)</code> [LSB]	<code>_Unwind_SetIP(GCC_3.0)</code> [LSB]	

2132

## 11.9 Data Definitions for libgcc\_s

2133 This section defines global identifiers and their values that are associated with  
2134 interfaces contained in libgcc\_s. These definitions are organized into groups that  
2135 correspond to system headers. This convention is used as a convenience for the  
2136 reader, and does not imply the existence of these headers, or their content. Where an  
2137 interface is defined as requiring a particular system header file all of the data  
2138 definitions for that system header file presented here shall be in effect.

2139 This section gives data definitions to promote binary application portability, not to  
2140 repeat source interface definitions available elsewhere. System providers and  
2141 application developers should use this ABI to supplement - not to replace - source  
2142 interface definition specifications.

2143 This specification uses the ISO C (1999) C Language as the reference programming  
2144 language, and data definitions are specified in ISO C format. The C language is used  
2145 here as a convenient notation. Using a C language description of these data objects  
2146 does not preclude their use by other programming languages.

### 11.9.1 unwind.h

```

2147 extern void _Unwind_DeleteException(struct _Unwind_Exception *);
2148 extern fde *_Unwind_Find_FDE(void *, struct dwarf_eh_base *);
2149 extern void _Unwind_DeleteException(struct _Unwind_Exception *);
2150 extern _Unwind_Ptr _Unwind_ForcedUnwind(struct _Unwind_Exception *,
2151                                         _Unwind_Stop_Fn, void *);
2152 extern _Unwind_Word _Unwind_GetGR(struct _Unwind_Context *, int);
2153 extern _Unwind_Ptr _Unwind_GetIP(struct _Unwind_Context *);
2154 extern _Unwind_Ptr _Unwind_GetLanguageSpecificData(struct
2155 _Unwind_Context
2156 *) ;
2157
```

```

2158     extern _Unwind_Ptr _Unwind_GetRegionStart(struct _Unwind_Context *);
2159     extern _Unwind_Reason_Code _Unwind_RaiseException(struct
2160     _Unwind_Exception
2161     *);
2162     extern void _Unwind_Resume(struct _Unwind_Exception *);
2163     extern void _Unwind_SetGR(struct _Unwind_Context *, int, u_int64_t);
2164     extern void _Unwind_SetIP(struct _Unwind_Context *, _Unwind_Ptr);
2165     extern void _Unwind_DeleteException(struct _Unwind_Exception *);
2166     extern fde *_Unwind_Find_FDE(void *, struct dwarf_eh_base *);
2167     extern _Unwind_Ptr _Unwind_ForcedUnwind(struct _Unwind_Exception *,
2168     _Unwind_Stop_Fn, void *);
2169     extern _Unwind_Ptr _Unwind_GetDataRelBase(struct _Unwind_Context *);
2170     extern _Unwind_Word _Unwind_GetGR(struct _Unwind_Context *, int);
2171     extern _Unwind_Ptr _Unwind_GetIP(struct _Unwind_Context *);
2172     extern _Unwind_Ptr _Unwind_GetLanguageSpecificData(struct
2173     _Unwind_Context
2174     *);
2175     extern _Unwind_Ptr _Unwind_GetRegionStart(struct _Unwind_Context *);
2176     extern _Unwind_Ptr _Unwind_GetTextRelBase(struct _Unwind_Context *);
2177     extern _Unwind_Reason_Code _Unwind_RaiseException(struct
2178     _Unwind_Exception
2179     *);
2180     extern void _Unwind_Resume(struct _Unwind_Exception *);
2181     extern void _Unwind_SetGR(struct _Unwind_Context *, int, u_int64_t);
2182     extern void _Unwind_SetIP(struct _Unwind_Context *, _Unwind_Ptr);
2183     extern _Unwind_Ptr _Unwind_ForcedUnwind(struct _Unwind_Exception *,
2184     _Unwind_Stop_Fn, void *);
2185     extern _Unwind_Ptr _Unwind_GetDataRelBase(struct _Unwind_Context *);
2186     extern _Unwind_Word _Unwind_GetGR(struct _Unwind_Context *, int);
2187     extern _Unwind_Ptr _Unwind_GetIP(struct _Unwind_Context *);
2188     extern _Unwind_Ptr _Unwind_GetLanguageSpecificData(struct
2189     _Unwind_Context
2190     *);
2191     extern _Unwind_Ptr _Unwind_GetRegionStart(struct _Unwind_Context *);
2192     extern _Unwind_Ptr _Unwind_GetTextRelBase(struct _Unwind_Context *);
2193     extern _Unwind_Reason_Code _Unwind_RaiseException(struct
2194     _Unwind_Exception
2195     *);
2196     extern void _Unwind_Resume(struct _Unwind_Exception *);
2197     extern void _Unwind_SetGR(struct _Unwind_Context *, int, u_int64_t);
2198     extern void _Unwind_SetIP(struct _Unwind_Context *, _Unwind_Ptr);
2199     extern void _Unwind_DeleteException(struct _Unwind_Exception *);
2200     extern fde *_Unwind_Find_FDE(void *, struct dwarf_eh_base *);
2201     extern _Unwind_Ptr _Unwind_ForcedUnwind(struct _Unwind_Exception *,
2202     _Unwind_Stop_Fn, void *);
2203     extern _Unwind_Ptr _Unwind_GetDataRelBase(struct _Unwind_Context *);
2204     extern _Unwind_Word _Unwind_GetGR(struct _Unwind_Context *, int);
2205     extern _Unwind_Ptr _Unwind_GetIP(struct _Unwind_Context *);
2206     extern _Unwind_Ptr _Unwind_GetLanguageSpecificData(struct
2207     _Unwind_Context
2208     *);
2209     extern _Unwind_Ptr _Unwind_GetRegionStart(struct _Unwind_Context *);
2210     extern _Unwind_Ptr _Unwind_GetTextRelBase(struct _Unwind_Context *);
2211     extern _Unwind_Reason_Code _Unwind_RaiseException(struct
2212     _Unwind_Exception
2213     *);
2214     extern void _Unwind_Resume(struct _Unwind_Exception *);
2215     extern void _Unwind_SetGR(struct _Unwind_Context *, int, u_int64_t);
2216     extern void _Unwind_SetIP(struct _Unwind_Context *, _Unwind_Ptr);
2217     extern void _Unwind_DeleteException(struct _Unwind_Exception *);
2218     extern fde *_Unwind_Find_FDE(void *, struct dwarf_eh_base *);
2219     extern _Unwind_Ptr _Unwind_ForcedUnwind(struct _Unwind_Exception *,
2220     _Unwind_Stop_Fn, void *);
2221     extern _Unwind_Ptr _Unwind_GetDataRelBase(struct _Unwind_Context *);

```

```

2222     extern _Unwind_Word _Unwind_GetGR(struct _Unwind_Context *, int);
2223     extern _Unwind_Ptr _Unwind_GetIP(struct _Unwind_Context *);
2224     extern _Unwind_Ptr _Unwind_GetLanguageSpecificData(struct
2225     _Unwind_Context
2226                                     *);
2227     extern _Unwind_Ptr _Unwind_GetRegionStart(struct _Unwind_Context *);
2228     extern _Unwind_Ptr _Unwind_GetTextRelBase(struct _Unwind_Context *);
2229     extern _Unwind_Reason_Code _Unwind_RaiseException(struct
2230     _Unwind_Exception
2231                                     *);
2232     extern void _Unwind_Resume(struct _Unwind_Exception *);
2233     extern void _Unwind_SetGR(struct _Unwind_Context *, int, u_int64_t);
2234     extern void _Unwind_SetIP(struct _Unwind_Context *, _Unwind_Ptr);
2235     extern void _Unwind_DeleteException(struct _Unwind_Exception *);
2236     extern fde *_Unwind_Find_FDE(void *, struct dwarf_eh_base *);
2237     extern _Unwind_Ptr _Unwind_ForcedUnwind(struct _Unwind_Exception *,
2238     _Unwind_Stop_Fn, void *);
2239     extern _Unwind_Ptr _Unwind_GetDataRelBase(struct _Unwind_Context *);
2240     extern _Unwind_Word _Unwind_GetGR(struct _Unwind_Context *, int);
2241     extern _Unwind_Ptr _Unwind_GetIP(struct _Unwind_Context *);
2242     extern _Unwind_Ptr _Unwind_GetLanguageSpecificData(void);
2243     extern _Unwind_Ptr _Unwind_GetRegionStart(struct _Unwind_Context *);
2244     extern _Unwind_Ptr _Unwind_GetTextRelBase(struct _Unwind_Context *);
2245     extern _Unwind_Reason_Code _Unwind_RaiseException(struct
2246     _Unwind_Exception
2247                                     *);
2248     extern void _Unwind_Resume(struct _Unwind_Exception *);
2249     extern void _Unwind_SetGR(struct _Unwind_Context *, int, u_int64_t);
2250     extern void _Unwind_SetIP(struct _Unwind_Context *, _Unwind_Ptr);
2251     extern void _Unwind_DeleteException(struct _Unwind_Exception *);
2252     extern fde *_Unwind_Find_FDE(void *, struct dwarf_eh_base *);
2253     extern _Unwind_Ptr _Unwind_ForcedUnwind(struct _Unwind_Exception *,
2254     _Unwind_Stop_Fn, void *);
2255     extern _Unwind_Ptr _Unwind_GetDataRelBase(struct _Unwind_Context *);
2256     extern _Unwind_Word _Unwind_GetGR(struct _Unwind_Context *, int);
2257     extern _Unwind_Ptr _Unwind_GetIP(struct _Unwind_Context *);
2258     extern _Unwind_Ptr _Unwind_GetLanguageSpecificData(void);
2259     extern _Unwind_Ptr _Unwind_GetRegionStart(struct _Unwind_Context *);
2260     extern _Unwind_Ptr _Unwind_GetTextRelBase(struct _Unwind_Context *);
2261     extern _Unwind_Reason_Code _Unwind_RaiseException(struct
2262     _Unwind_Exception
2263                                     *);
2264     extern void _Unwind_Resume(struct _Unwind_Exception *);
2265     extern void _Unwind_SetGR(struct _Unwind_Context *, int, u_int64_t);
2266     extern void _Unwind_SetIP(struct _Unwind_Context *, _Unwind_Ptr);
2267     extern _Unwind_Reason_Code _Unwind_Backtrace(_Unwind_Trace_Fn, void
2268     *);
2269     extern _Unwind_Reason_Code _Unwind_Backtrace(_Unwind_Trace_Fn, void
2270     *);
2271     extern _Unwind_Reason_Code _Unwind_Backtrace(_Unwind_Trace_Fn, void
2272     *);
2273     extern _Unwind_Reason_Code _Unwind_Backtrace(_Unwind_Trace_Fn, void
2274     *);
2275     extern _Unwind_Reason_Code _Unwind_Backtrace(_Unwind_Trace_Fn, void
2276     *);
2277     extern _Unwind_Reason_Code _Unwind_Backtrace(_Unwind_Trace_Fn, void
2278     *);
2279     extern _Unwind_Reason_Code _Unwind_Backtrace(_Unwind_Trace_Fn, void
2280     *);
2281     extern _Unwind_Reason_Code _Unwind_GetCFA(struct _Unwind_Context *);
2282     extern _Unwind_Reason_Code _Unwind_GetCFA(struct _Unwind_Context *);
2283     extern _Unwind_Reason_Code _Unwind_GetCFA(struct _Unwind_Context *);
2284     extern _Unwind_Reason_Code _Unwind_GetCFA(struct _Unwind_Context *);
2285     extern _Unwind_Reason_Code _Unwind_GetCFA(struct _Unwind_Context *);

```

```

2286     extern _Unwind_Reason_Code _Unwind_GetCFA(struct _Unwind_Context *);
2287     extern _Unwind_Reason_Code _Unwind_GetCFA(struct _Unwind_Context *);
2288     extern _Unwind_Reason_Code _Unwind_Resume_or_Rethrow(struct
2289
2290         _Unwind_Exception *);
2291     extern _Unwind_Reason_Code _Unwind_Resume_or_Rethrow(struct
2292
2293         _Unwind_Exception *);
2294     extern _Unwind_Reason_Code _Unwind_Resume_or_Rethrow(struct
2295
2296         _Unwind_Exception *);
2297     extern _Unwind_Reason_Code _Unwind_Resume_or_Rethrow(struct
2298
2299         _Unwind_Exception *);
2300     extern _Unwind_Reason_Code _Unwind_Resume_or_Rethrow(struct
2301
2302         _Unwind_Exception *);
2303     extern _Unwind_Reason_Code _Unwind_Resume_or_Rethrow(struct
2304
2305         _Unwind_Exception *);
2306     extern _Unwind_Reason_Code _Unwind_Resume_or_Rethrow(struct
2307
2308         _Unwind_Exception *);
2309     extern void *_Unwind_FindEnclosingFunction(void *);
2310     extern void *_Unwind_FindEnclosingFunction(void *);
2311     extern void *_Unwind_FindEnclosingFunction(void *);
2312     extern void *_Unwind_FindEnclosingFunction(void *);
2313     extern void *_Unwind_FindEnclosingFunction(void *);
2314     extern void *_Unwind_FindEnclosingFunction(void *);
2315     extern void *_Unwind_FindEnclosingFunction(void *);
2316     extern _Unwind_Word _Unwind_GetBSP(struct _Unwind_Context *);

```

## 11.10 Interface Definitions for libgcc\_s

2317 The interfaces defined on the following pages are included in libgcc\_s and are  
2318 defined by this specification. Unless otherwise noted, these interfaces shall be  
2319 included in the source standard.

2320 Other interfaces listed in Section 11.8 shall behave as described in the referenced  
2321 base document.

### **`_Unwind_DeleteException`**

#### **Name**

2322 `_Unwind_DeleteException` – private C++ error handling method

#### **Synopsis**

2323 `void _Unwind_DeleteException(struct _Unwind_Exception * object);`

#### **Description**

2324 `_Unwind_DeleteException()` deletes the given exception *object*. If a given  
2325 runtime resumes normal execution after catching a foreign exception, it will not  
2326 know how to delete that exception. Such an exception shall be deleted by calling  
2327 `_Unwind_DeleteException()`. This is a convenience function that calls the function  
2328 pointed to by the *exception\_cleanup* field of the exception header.

## **`_Unwind_ForcedUnwind`**

### **Name**

2329 `_Unwind_ForcedUnwind` — private C++ error handling method

### **Synopsis**

```
2330 _Unwind_Reason_Code _Unwind_ForcedUnwind(struct _Unwind_Exception *  
2331 object, _Unwind_Stop_Fn stop, void * stop_parameter);
```

### **Description**

2332 `_Unwind_ForcedUnwind()` raises an exception for forced unwinding, passing along  
2333 the given exception *object*, which should have its *exception\_class* and  
2334 *exception\_cleanup* fields set. The exception *object* has been allocated by the  
2335 language-specific runtime, and has a language-specific format, except that it shall  
2336 contain an `_Unwind_Exception` struct.

2337 Forced unwinding is a single-phase process. *stop* and *stop\_parameter* control the  
2338 termination of the unwind process instead of the usual personality routine query.  
2339 *stop* is called for each unwind frame, with the parameteres described for the usual  
2340 personality routine below, plus an additional *stop\_parameter*.

### **Return Value**

2341 When *stop* identifies the destination frame, it transfers control to the user code as  
2342 appropriate without returning, normally after calling `_Unwind_DeleteException()`.  
2343 If not, then it should return an `_Unwind_Reason_Code` value.

2344 If *stop* returns any reason code other than `_URC_NO_REASON`, then the stack state is  
2345 indeterminate from the point of view of the caller of `_Unwind_ForcedUnwind()`.  
2346 Rather than attempt to return, therefore, the unwind library should use the  
2347 *exception\_cleanup* entry in the exception, and then call `abort()`.

2348 `_URC_NO_REASON`

2349 This is not the destination from. The unwind runtime will call frame's  
2350 personality routine with the `_UA_FORCE_UNWIND` and `_UA_CLEANUP_PHASE` flag  
2351 set in *actions*, and then unwind to the next frame and call the `stop()` function  
2352 again.

2353 `_URC_END_OF_STACK`

2354 In order to allow `_Unwind_ForcedUnwind()` to perform special processing  
2355 when it reaches the end of the stack, the unwind runtime will call it after the last  
2356 frame is rejected, with a NULL stack pointer in the context, and the `stop()`  
2357 function shall catch this condition. It may return this code if it cannot handle  
2358 end-of-stack.

2359 `_URC_FATAL_PHASE2_ERROR`

2360 The `stop()` function may return this code for other fatal conditions like stack  
2361 corruption.

## **`_Unwind_GetGR`**

### **Name**

2362 `_Unwind_GetGR` – private C++ error handling method

### **Synopsis**

2363 `_Unwind_Word _Unwind_GetGR(struct _Unwind_Context * context, int index);`

### **Description**

2364 `_Unwind_GetGR()` returns data at *index* found in *context*. The register is identified  
2365 by its index: 0 to 31 are for the fixed registers, and 32 to 127 are for the stacked  
2366 registers.

2367 During the two phases of unwinding, only GR1 has a guaranteed value, which is the  
2368 global pointer of the frame referenced by the unwind *context*. If the register has its  
2369 NAT bit set, the behavior is unspecified.

## **`_Unwind_GetIP`**

### **Name**

2370 `_Unwind_GetIP` – private C++ error handling method

### **Synopsis**

2371 `_Unwind_Ptr _Unwind_GetIP(struct _Unwind_Context * context);`

### **Description**

2372 `_Unwind_GetIP()` returns the instruction pointer value for the routine identified by  
2373 the unwind *context*.

## **`_Unwind_GetLanguageSpecificData`**

### **Name**

2374 `_Unwind_GetLanguageSpecificData` – private C++ error handling method

### **Synopsis**

2375 `_Unwind_Ptr _Unwind_GetLanguageSpecificData(struct _Unwind_Context *`  
2376 `context, uint value);`

### **Description**

2377 `_Unwind_GetLanguageSpecificData()` returns the address of the language specific  
2378 data area for the current stack frame.

## **`_Unwind_GetRegionStart`**

### **Name**

2379 `_Unwind_GetRegionStart` – private C++ error handling method

### **Synopsis**

2380 `_Unwind_Ptr _Unwind_GetRegionStart(struct _Unwind_Context * context);`

### **Description**

2381 `_Unwind_GetRegionStart()` routine returns the address (i.e., 0) of the beginning of  
2382 the procedure or code fragment described by the current unwind descriptor block.

## **`_Unwind_RaiseException`**

### **Name**

2383 `_Unwind_RaiseException` – private C++ error handling method

### **Synopsis**

2384 `_Unwind_Reason_Code _Unwind_RaiseException(struct _Unwind_Exception *  
2385 object);`

### **Description**

2386 `_Unwind_RaiseException()` raises an exception, passing along the given exception  
2387 *object*, which should have its *exception\_class* and *exception\_cleanup* fields set.  
2388 The exception object has been allocated by the language-specific runtime, and has a  
2389 language-specific format, exception that it shall contain an `_Unwind_Exception`.

### **Return Value**

2390 `_Unwind_RaiseException()` does not return unless an error condition is found. If  
2391 an error condition occurs, an `_Unwind_Reason_Code` is returned:

2392 `_URC_END_OF_STACK`

2393 The unwinder encountered the end of the stack during phase one without  
2394 finding a handler. The unwind runtime will not have modified the stack. The  
2395 C++ runtime will normally call `uncaught_exception()` in this case.

2396 `_URC_FATAL_PHASE1_ERROR`

2397 The unwinder encountered an unexpected error during phase one, because of  
2398 something like stack corruption. The unwind runtime will not have modified  
2399 the stack. The C++ runtime will normally call `terminate()` in this case.

2400 `_URC_FATAL_PHASE2_ERROR`

2401 The unwinder encountered an unexpected error during phase two. This is  
2402 usually a *throw*, which will call `terminate()`.



## **\_Unwind\_Resume**

### **Name**

2403 `_Unwind_Resume` – private C++ error handling method

### **Synopsis**

2404 `void _Unwind_Resume(struct _Unwind_Exception * object);`

### **Description**

2405 `_Unwind_Resume()` resumes propagation of an existing exception *object*. A call to  
2406 this routine is inserted as the end of a landing pad that performs cleanup, but does  
2407 not resume normal execution. It causes unwinding to proceed further.

## **\_Unwind\_SetGR**

### **Name**

2408 `_Unwind_SetGR` – private C++ error handling method

### **Synopsis**

2409 `void _Unwind_SetGR(struct _Unwind_Context * context, int index, uint value);`

### **Description**

2410 `_Unwind_SetGR()` sets the *value* of the register *indexed* for the routine identified by  
2411 the unwind *context*.

## **\_Unwind\_SetIP**

### **Name**

2412 `_Unwind_SetIP` – private C++ error handling method

### **Synopsis**

2413 `void _Unwind_SetIP(struct _Unwind_Context * context, uint value);`

### **Description**

2414 `_Unwind_SetIP()` sets the *value* of the instruction pointer for the routine identified  
2415 by the unwind *context*.

## **11.11 Interfaces for libdl**

2416 Table 11-33 defines the library name and shared object name for the libdl library

2417 **Table 11-33 libdl Definition**

Library:	libdl
SONAME:	libdl.so.2

2418

2419 The behavior of the interfaces in this library is specified by the following specifica-  
2420 tions:

2421 [LSB] This Specification  
[SUSv3] ISO POSIX (2003)

### 11.11.1 Dynamic Loader

#### 11.11.1.1 Interfaces for Dynamic Loader

2423 An LSB conforming implementation shall provide the architecture specific functions  
2424 for Dynamic Loader specified in Table 11-34, with the full mandatory functionality  
2425 as described in the referenced underlying specification.

2426 **Table 11-34 libdl - Dynamic Loader Function Interfaces**

dladdr(GLIBC_2.0) [LSB]	dlclose(GLIBC_2.0) [SUSv3]	dLError(GLIBC_2.0) [SUSv3]	dlopen(GLIBC_2.1) [LSB]
dlsym(GLIBC_2.0) [LSB]			

2427

### 11.12 Data Definitions for libdl

2428 This section defines global identifiers and their values that are associated with  
2429 interfaces contained in libdl. These definitions are organized into groups that  
2430 correspond to system headers. This convention is used as a convenience for the  
2431 reader, and does not imply the existence of these headers, or their content. Where an  
2432 interface is defined as requiring a particular system header file all of the data  
2433 definitions for that system header file presented here shall be in effect.

2434 This section gives data definitions to promote binary application portability, not to  
2435 repeat source interface definitions available elsewhere. System providers and  
2436 application developers should use this ABI to supplement - not to replace - source  
2437 interface definition specifications.

2438 This specification uses the ISO C (1999) C Language as the reference programming  
2439 language, and data definitions are specified in ISO C format. The C language is used  
2440 here as a convenient notation. Using a C language description of these data objects  
2441 does not preclude their use by other programming languages.

#### 11.12.1 dlfcn.h

```
2442 extern int dladdr(const void *, Dl_info *);
2443 extern int dlclose(void *);
2444 extern char *dLError(void);
2445 extern void *dlopen(char *, int);
2446 extern void *dlsym(void *, char *);
2447
```

### 11.13 Interfaces for libcrypt

2448 Table 11-35 defines the library name and shared object name for the libcrypt library

2449 **Table 11-35 libcrypt Definition**

Library:	libcrypt
SONAME:	libcrypt.so.1

2450

2451 The behavior of the interfaces in this library is specified by the following specifica-  
 2452 tions:

2453 [SUSv3] ISO POSIX (2003)

## 11.13.1 Encryption

### 2454 11.13.1.1 Interfaces for Encryption

2455 An LSB conforming implementation shall provide the architecture specific functions  
 2456 for Encryption specified in Table 11-36, with the full mandatory functionality as  
 2457 described in the referenced underlying specification.

2458 **Table 11-36 libcrypt - Encryption Function Interfaces**

crypt(GLIBC_2.0) [SUSv3]	encrypt(GLIBC_2. 0) [SUSv3]	setkey(GLIBC_2.0 ) [SUSv3]	
-----------------------------	--------------------------------	-------------------------------	--

2459

## **IV Utility Libraries**

## 12 Libraries

1           An LSB-conforming implementation shall also support some utility libraries which  
2           are built on top of the interfaces provided by the base libraries. These libraries  
3           implement common functionality, and hide additional system dependent  
4           information such as file formats and device names.

### 12.1 Interfaces for libz

5           Table 12-1 defines the library name and shared object name for the libz library

6           **Table 12-1 libz Definition**

Library:	libz
SONAME:	libz.so.1

7

#### 12.1.1 Compression Library

##### 8           12.1.1.1 Interfaces for Compression Library

9           No external functions are defined for libz - Compression Library in this part of the  
10          specification. See also the generic specification.

### 12.2 Data Definitions for libz

11          This section defines global identifiers and their values that are associated with  
12          interfaces contained in libz. These definitions are organized into groups that  
13          correspond to system headers. This convention is used as a convenience for the  
14          reader, and does not imply the existence of these headers, or their content. Where an  
15          interface is defined as requiring a particular system header file all of the data  
16          definitions for that system header file presented here shall be in effect.

17          This section gives data definitions to promote binary application portability, not to  
18          repeat source interface definitions available elsewhere. System providers and  
19          application developers should use this ABI to supplement - not to replace - source  
20          interface definition specifications.

21          This specification uses the ISO C (1999) C Language as the reference programming  
22          language, and data definitions are specified in ISO C . The C language is used here  
23          as a convenient notation. Using a C language description of these data objects does  
24          not preclude their use by other programming languages.

#### 12.2.1 zlib.h

```
25  
26       extern int gzread(gzFile, voidp, unsigned int);  
27       extern int gzclose(gzFile);  
28       extern gzFile gzopen(const char *, const char *);  
29       extern gzFile gzdopen(int, const char *);  
30       extern int gzwrite(gzFile, voidpc, unsigned int);  
31       extern int gzflush(gzFile, int);  
32       extern const char *gzerror(gzFile, int *);  
33       extern uLong Adler32(uLong, const Bytef *, uInt);  
34       extern int compress(Bytef *, uLongf *, const Bytef *, uLong);  
35       extern int compress2(Bytef *, uLongf *, const Bytef *, uLong, int);  
36       extern uLong crc32(uLong, const Bytef *, uInt);  
37       extern int deflate(z_streamp, int);
```

```

38     extern int deflateCopy(z_streamp, z_streamp);
39     extern int deflateEnd(z_streamp);
40     extern int deflateInit2_(z_streamp, int, int, int, int, int, const char
41     *,
42         int);
43     extern int deflateInit_(z_streamp, int, const char *, int);
44     extern int deflateParams(z_streamp, int, int);
45     extern int deflateReset(z_streamp);
46     extern int deflateSetDictionary(z_streamp, const Bytef *, uInt);
47     extern const uLongf *get_crc_table(void);
48     extern int gzeof(gzFile);
49     extern int gzgetc(gzFile);
50     extern char *gzgets(gzFile, char *, int);
51     extern int gzprintf(gzFile, const char *, ...);
52     extern int gzputc(gzFile, int);
53     extern int gzputs(gzFile, const char *);
54     extern int gzrewind(gzFile);
55     extern z_off_t gzseek(gzFile, z_off_t, int);
56     extern int gzsetparams(gzFile, int, int);
57     extern z_off_t gztell(gzFile);
58     extern int inflate(z_streamp, int);
59     extern int inflateEnd(z_streamp);
60     extern int inflateInit2_(z_streamp, int, const char *, int);
61     extern int inflateInit_(z_streamp, const char *, int);
62     extern int inflateReset(z_streamp);
63     extern int inflateSetDictionary(z_streamp, const Bytef *, uInt);
64     extern int inflateSync(z_streamp);
65     extern int inflateSyncPoint(z_streamp);
66     extern int uncompress(Bytef *, uLongf *, const Bytef *, uLong);
67     extern const char *zError(int);
68     extern const char *zlibVersion(void);
69     extern uLong deflateBound(z_streamp, uLong);
70     extern uLong compressBound(uLong);

```

## 12.3 Interfaces for libncurses

71 Table 12-2 defines the library name and shared object name for the libncurses library

72 **Table 12-2 libncurses Definition**

73 Library:	libncurses
SONAME:	libncurses.so.5

### 12.3.1 Curses

#### 74 12.3.1.1 Interfaces for Curses

75 No external functions are defined for libncurses - Curses in this part of the  
76 specification. See also the generic specification.

## 12.4 Data Definitions for libncurses

77 This section defines global identifiers and their values that are associated with  
78 interfaces contained in libncurses. These definitions are organized into groups that  
79 correspond to system headers. This convention is used as a convenience for the  
80 reader, and does not imply the existence of these headers, or their content. Where an  
81 interface is defined as requiring a particular system header file all of the data  
82 definitions for that system header file presented here shall be in effect.

83 This section gives data definitions to promote binary application portability, not to  
 84 repeat source interface definitions available elsewhere. System providers and  
 85 application developers should use this ABI to supplement - not to replace - source  
 86 interface definition specifications.

87 This specification uses the ISO C (1999) C Language as the reference programming  
 88 language, and data definitions are specified in ISO C. The C language is used here  
 89 as a convenient notation. Using a C language description of these data objects does  
 90 not preclude their use by other programming languages.

## 12.4.1 curses.h

```

91
92 extern int addch(const chtype);
93 extern int addchnstr(const chtype *, int);
94 extern int addchstr(const chtype *);
95 extern int addnstr(const char *, int);
96 extern int addstr(const char *);
97 extern int attroff(int);
98 extern int attron(int);
99 extern int attrset(int);
100 extern int attr_get(attr_t *, short *, void *);
101 extern int attr_off(attr_t, void *);
102 extern int attr_on(attr_t, void *);
103 extern int attr_set(attr_t, short, void *);
104 extern int baudrate(void);
105 extern int beep(void);
106 extern int bkgd(chtype);
107 extern void bkgdset(chtype);
108 extern int border(chtype, chtype, chtype, chtype, chtype, chtype,
109 chtype,
110 chtype);
111 extern int box(WINDOW *, chtype, chtype);
112 extern bool can_change_color(void);
113 extern int cbreak(void);
114 extern int chgat(int, attr_t, short, const void *);
115 extern int clear(void);
116 extern int clearok(WINDOW *, bool);
117 extern int clrtoeol(void);
118 extern int clrtoeol(void);
119 extern int color_content(short, short *, short *, short *);
120 extern int color_set(short, void *);
121 extern int copywin(const WINDOW *, WINDOW *, int, int, int, int, int,
122 int,
123 int);
124 extern int curs_set(int);
125 extern int def_prog_mode(void);
126 extern int def_shell_mode(void);
127 extern int delay_output(int);
128 extern int delch(void);
129 extern void delscreen(SCREEN *);
130 extern int delwin(WINDOW *);
131 extern int deleteln(void);
132 extern WINDOW *derwin(WINDOW *, int, int, int, int);
133 extern int doupdate(void);
134 extern WINDOW *dupwin(WINDOW *);
135 extern int echo(void);
136 extern int echochar(const chtype);
137 extern int erase(void);
138 extern int endwin(void);
139 extern char erasechar(void);
140 extern void filter(void);
141 extern int flash(void);

```

```

142     extern int flushing(void);
143     extern chtype getbkgd(WINDOW *);
144     extern int getch(void);
145     extern int getnstr(char *, int);
146     extern int getstr(char *);
147     extern WINDOW *getwin(FILE *);
148     extern int halfdelay(int);
149     extern bool has_colors(void);
150     extern bool has_ic(void);
151     extern bool has_il(void);
152     extern int hline(chtype, int);
153     extern void idcok(WINDOW *, bool);
154     extern int idlok(WINDOW *, bool);
155     extern void immedok(WINDOW *, bool);
156     extern chtype inch(void);
157     extern int inchnstr(chtype *, int);
158     extern int inchstr(chtype *);
159     extern WINDOW *initscr(void);
160     extern int init_color(short, short, short, short);
161     extern int init_pair(short, short, short);
162     extern int innstr(char *, int);
163     extern int insch(chtype);
164     extern int insdelln(int);
165     extern int insertln(void);
166     extern int insnstr(const char *, int);
167     extern int insstr(const char *);
168     extern int instr(char *);
169     extern int intrflush(WINDOW *, bool);
170     extern bool isendwin(void);
171     extern bool is_linetouched(WINDOW *, int);
172     extern bool is_wintouched(WINDOW *);
173     extern const char *keyname(int);
174     extern int keypad(WINDOW *, bool);
175     extern char killchar(void);
176     extern int leaveok(WINDOW *, bool);
177     extern char *longname(void);
178     extern int meta(WINDOW *, bool);
179     extern int move(int, int);
180     extern int mvaddch(int, int, const chtype);
181     extern int mvaddchnstr(int, int, const chtype *, int);
182     extern int mvaddchstr(int, int, const chtype *);
183     extern int mvaddnstr(int, int, const char *, int);
184     extern int mvaddstr(int, int, const char *);
185     extern int mvchgat(int, int, int, attr_t, short, const void *);
186     extern int mvcur(int, int, int, int);
187     extern int mvdelch(int, int);
188     extern int mvderwin(WINDOW *, int, int);
189     extern int mvgetch(int, int);
190     extern int mvgetnstr(int, int, char *, int);
191     extern int mvgetstr(int, int, char *);
192     extern int mvhline(int, int, chtype, int);
193     extern chtype mvinch(int, int);
194     extern int mvinchnstr(int, int, chtype *, int);
195     extern int mvinchstr(int, int, chtype *);
196     extern int mvinnstr(int, int, char *, int);
197     extern int mvinsch(int, int, chtype);
198     extern int mvinsnstr(int, int, const char *, int);
199     extern int mvinsstr(int, int, const char *);
200     extern int mvinstr(int, int, char *);
201     extern int mvprintw(int, int, char *, ...);
202     extern int mvscanw(int, int, const char *, ...);
203     extern int mvvline(int, int, chtype, int);
204     extern int mvwaddch(WINDOW *, int, int, const chtype);
205     extern int mvwaddchnstr(WINDOW *, int, int, const chtype *, int);

```



```

206     extern int mvwaddchstr(WINDOW *, int, int, const chtype *);
207     extern int mvwaddnstr(WINDOW *, int, int, const char *, int);
208     extern int mvwaddstr(WINDOW *, int, int, const char *);
209     extern int mvwchgat(WINDOW *, int, int, int, attr_t, short, const void
210     *);
211     extern int mvwdelch(WINDOW *, int, int);
212     extern int mvwgetch(WINDOW *, int, int);
213     extern int mvwgetnstr(WINDOW *, int, int, char *, int);
214     extern int mvwgetstr(WINDOW *, int, int, char *);
215     extern int mvwhline(WINDOW *, int, int, chtype, int);
216     extern int mvwin(WINDOW *, int, int);
217     extern chtype mvwinch(WINDOW *, int, int);
218     extern int mvwinchnstr(WINDOW *, int, int, chtype *, int);
219     extern int mvwinchstr(WINDOW *, int, int, chtype *);
220     extern int mvwinnstr(WINDOW *, int, int, char *, int);
221     extern int mvwinsch(WINDOW *, int, int, chtype);
222     extern int mvwinsnstr(WINDOW *, int, int, const char *, int);
223     extern int mvwinsstr(WINDOW *, int, int, const char *);
224     extern int mvwinstr(WINDOW *, int, int, char *);
225     extern int mvwprintw(WINDOW *, int, int, char *, ...);
226     extern int mvwscanw(WINDOW *, int, int, const char *, ...);
227     extern int mvwvline(WINDOW *, int, int, chtype, int);
228     extern int napms(int);
229     extern WINDOW *newpad(int, int);
230     extern SCREEN *newterm(const char *, FILE *, FILE *);
231     extern WINDOW *newwin(int, int, int, int);
232     extern int nl(void);
233     extern int nocbreak(void);
234     extern int nodelay(WINDOW *, bool);
235     extern int noecho(void);
236     extern int nonl(void);
237     extern void noqiflush(void);
238     extern int noraw(void);
239     extern int notimeout(WINDOW *, bool);
240     extern int overlay(const WINDOW *, WINDOW *);
241     extern int overwrite(const WINDOW *, WINDOW *);
242     extern int pair_content(short, short *, short *);
243     extern int pechochar(WINDOW *, chtype);
244     extern int pnoutrefresh(WINDOW *, int, int, int, int, int, int);
245     extern int prefresh(WINDOW *, int, int, int, int, int, int);
246     extern int printw(char *, ...);
247     extern int putwin(WINDOW *, FILE *);
248     extern void qiflush(void);
249     extern int raw(void);
250     extern int redrawwin(WINDOW *);
251     extern int refresh(void);
252     extern int resetty(void);
253     extern int reset_prog_mode(void);
254     extern int reset_shell_mode(void);
255     extern int ripoffline(int, int (*init) (WINDOW *, int)
256     );
257     extern int savetty(void);
258     extern int scanw(const char *, ...);
259     extern int scr_dump(const char *);
260     extern int scr_init(const char *);
261     extern int scrl(int);
262     extern int scroll(WINDOW *);
263     extern int scrollok(WINDOW *, typedef unsigned char bool);
264     extern int scr_restore(const char *);
265     extern int scr_set(const char *);
266     extern int setscreg(int, int);
267     extern SCREEN *set_term(SCREEN *);
268     extern int slk_attroff(const typedef unsigned long int chtype);
269     extern int slk_attron(const typedef unsigned long int chtype);

```

```

270     extern int slk_attrset(const typedef unsigned long int chtype);
271     extern int slk_attr_set(const typedef chtype attr_t, short, void *);
272     extern int slk_clear(void);
273     extern int slk_color(short);
274     extern int slk_init(int);
275     extern char *slk_label(int);
276     extern int slk_noutrefresh(void);
277     extern int slk_refresh(void);
278     extern int slk_restore(void);
279     extern int slk_set(int, const char *, int);
280     extern int slk_touch(void);
281     extern int standout(void);
282     extern int standend(void);
283     extern int start_color(void);
284     extern WINDOW *subpad(WINDOW *, int, int, int, int);
285     extern WINDOW *subwin(WINDOW *, int, int, int, int);
286     extern int syncok(WINDOW *, typedef unsigned char bool);
287     extern typedef unsigned long int chtype termattrs(void);
288     extern char *termname(void);
289     extern void timeout(int);
290     extern int typeahead(int);
291     extern int ungetch(int);
292     extern int untouchwin(WINDOW *);
293     extern void use_env(typedef unsigned char bool);
294     extern int vidattr(typedef unsigned long int chtype);
295     extern int vidputs(typedef unsigned long int chtype,
296                       int (*vidputs_int) (int)
297                       );
298     extern int vline(typedef unsigned long int chtype, int);
299     extern int vwprintw(WINDOW *, char *, typedef void *va_list);
300     extern int vw_printw(WINDOW *, const char *, typedef void *va_list);
301     extern int vwscanw(WINDOW *, const char *, typedef void *va_list);
302     extern int vw_scanw(WINDOW *, const char *, typedef void *va_list);
303     extern int waddch(WINDOW *, const typedef unsigned long int chtype);
304     extern int waddchnstr(WINDOW *, const typedef unsigned long int chtype
305                          *,
306                          int);
307     extern int waddchstr(WINDOW *, const typedef unsigned long int chtype
308                          *);
309     extern int waddnstr(WINDOW *, const char *, int);
310     extern int waddstr(WINDOW *, const char *);
311     extern int wattron(WINDOW *, int);
312     extern int wattroff(WINDOW *, int);
313     extern int wattrset(WINDOW *, int);
314     extern int wattr_get(WINDOW *, attr_t *, short *, void *);
315     extern int wattr_on(WINDOW *, typedef chtype attr_t, void *);
316     extern int wattr_off(WINDOW *, typedef chtype attr_t, void *);
317     extern int wattr_set(WINDOW *, typedef chtype attr_t, short, void *);
318     extern int wbkgd(WINDOW *, typedef unsigned long int chtype);
319     extern void wbkgdset(WINDOW *, typedef unsigned long int chtype);
320     extern int wborder(WINDOW *, typedef unsigned long int chtype,
321                       typedef unsigned long int chtype,
322                       typedef unsigned long int chtype,
323                       typedef unsigned long int chtype,
324                       typedef unsigned long int chtype,
325                       typedef unsigned long int chtype,
326                       typedef unsigned long int chtype,
327                       typedef unsigned long int chtype);
328     extern int wchgat(WINDOW *, int, typedef chtype attr_t, short,
329                      const void *);
330     extern int wclear(WINDOW *);
331     extern int wclrtoeol(WINDOW *);
332     extern int wclrtoeol(WINDOW *);
333     extern int wcolor_set(WINDOW *, short, void *);

```

```

334     extern void wcuryncup(WINDOW *);
335     extern int wdelch(WINDOW *);
336     extern int wdeleteln(WINDOW *);
337     extern int wechochar(WINDOW *, const typedef unsigned long int chtype);
338     extern int werase(WINDOW *);
339     extern int wgetch(WINDOW *);
340     extern int wgetnstr(WINDOW *, char *, int);
341     extern int wgetstr(WINDOW *, char *);
342     extern int whline(WINDOW *, typedef unsigned long int chtype, int);
343     extern typedef unsigned long int chtype winch(WINDOW *);
344     extern int winchnstr(WINDOW *, chtype *, int);
345     extern int winchstr(WINDOW *, chtype *);
346     extern int winnstr(WINDOW *, char *, int);
347     extern int winsch(WINDOW *, typedef unsigned long int chtype);
348     extern int winsdelln(WINDOW *, int);
349     extern int winsertln(WINDOW *);
350     extern int winsnstr(WINDOW *, const char *, int);
351     extern int winsstr(WINDOW *, const char *);
352     extern int winstr(WINDOW *, char *);
353     extern int wmove(WINDOW *, int, int);
354     extern int wnoutrefresh(WINDOW *);
355     extern int wprintw(WINDOW *, char *, ...);
356     extern int wredrawln(WINDOW *, int, int);
357     extern int wrefresh(WINDOW *);
358     extern int wscanw(WINDOW *, const char *, ...);
359     extern int wscrl(WINDOW *, int);
360     extern int wsetscrreg(WINDOW *, int, int);
361     extern int wstandout(WINDOW *);
362     extern int wstandend(WINDOW *);
363     extern void wsyncdown(WINDOW *);
364     extern void wsyncup(WINDOW *);
365     extern void wtimeout(WINDOW *, int);
366     extern int wtouchln(WINDOW *, int, int, int);
367     extern int wvline(WINDOW *, typedef unsigned long int chtype, int);
368     extern char *unctrl(typedef unsigned long int chtype);
369     extern int COLORS(void);
370     extern int COLOR_PAIRS(void);
371     extern chtype acs_map(void);
372     extern WINDOW *curscr(void);
373     extern WINDOW *stdscr(void);
374     extern int COLS(void);
375     extern int LINES(void);
376     extern int touchline(WINDOW *, int, int);
377     extern int touchwin(WINDOW *);

```

## 12.4.2 term.h

```

378
379     extern int putp(const char *);
380     extern int tigetflag(const char *);
381     extern int tigetnum(const char *);
382     extern char *tigetstr(const char *);
383     extern char *tparm(const char *, ...);
384     extern TERMINAL *set_curterm(TERMINAL *);
385     extern int del_curterm(TERMINAL *);
386     extern int restartterm(char *, int, int *);
387     extern int setupterm(char *, int, int *);
388     extern char *tgetstr(char *, char **);
389     extern char *tgoto(const char *, int, int);
390     extern int tgetent(char *, const char *);
391     extern int tgetflag(char *);
392     extern int tgetnum(char *);
393     extern int tputs(const char *, int, int (*putcproc) (int)
394         );

```

395           extern TERMINAL \*cur\_term(void);

## 12.5 Interfaces for libutil

396           Table 12-3 defines the library name and shared object name for the libutil library

397           **Table 12-3 libutil Definition**

Library:	libutil
SONAME:	libutil.so.1

398

399           The behavior of the interfaces in this library is specified by the following specifica-  
400           tions:

401           [LSB] This Specification

### 12.5.1 Utility Functions

#### 12.5.1.1 Interfaces for Utility Functions

402           An LSB conforming implementation shall provide the architecture specific functions  
403           for Utility Functions specified in Table 12-4, with the full mandatory functionality as  
404           described in the referenced underlying specification.  
405

406           **Table 12-4 libutil - Utility Functions Function Interfaces**

forkpty(GLIBC_2.0) [LSB]	login(GLIBC_2.0) [LSB]	login_tty(GLIBC_2.0) [LSB]	logout(GLIBC_2.0) [LSB]
logwtmp(GLIBC_2.0) [LSB]	openpty(GLIBC_2.0) [LSB]		

407

## **V Package Format and Installation**

## 13 Software Installation

### 13.1 Package Dependencies

1           The LSB runtime environment shall provide the following dependencies.

2           `lsb-core-ia64`

3           This dependency is used to indicate that the application is dependent on  
4           features contained in the LSB-Core specification.

5           These dependencies shall have a version of 3.0.

6           Other LSB modules may add additional dependencies; such dependencies shall  
7           have the format `lsb-module-ia64`.

### 13.2 Package Architecture Considerations

8           All packages must specify an architecture of `IA64`. A LSB runtime environment must  
9           accept an architecture of `ia64` even if the native architecture is different.

10          The `archnum` value in the Lead Section shall be `0x0009`.

## Annex A Alphabetical Listing of Interfaces

### A.1 libgcc\_s

1           The behavior of the interfaces in this library is specified by the following Standards.  
2           This Specification [LSB]

3           **Table A-1 libgcc\_s Function Interfaces**

_Unwind_Backtrace[LSB]	_Unwind_GetCFA[LSB]	_Unwind_RaiseException[LSB]
_Unwind_DeleteException[LSB]	_Unwind_GetGR[LSB]	_Unwind_Resume[LSB]
_Unwind_FindEnclosingFunction[LSB]	_Unwind_GetIP[LSB]	_Unwind_Resume_or_Rethrow[LSB]
_Unwind_ForcedUnwind[LSB]	_Unwind_GetLanguageSpecificData[LSB]	_Unwind_SetGR[LSB]
_Unwind_GetBSP[LSB]	_Unwind_GetRegionStart[LSB]	_Unwind_SetIP[LSB]

4

### A.2 libm

5           The behavior of the interfaces in this library is specified by the following Standards.  
6           ISO C (1999) [ISOC99]  
7           ISO POSIX (2003) [SUSv3]

7           **Table A-2 libm Function Interfaces**

__fpclassify[ISOC99]	__signbit[ISOC99]	exp2l[SUSv3]
----------------------	-------------------	--------------

8

## Annex B GNU Free Documentation License (Informative)

1           This specification is published under the terms of the GNU Free Documentation  
2 License, Version 1.1, March 2000

3           Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston,  
4 MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of  
5 this license document, but changing it is not allowed.

### B.1 PREAMBLE

6           The purpose of this License is to make a manual, textbook, or other written  
7 document "free" in the sense of freedom: to assure everyone the effective freedom to  
8 copy and redistribute it, with or without modifying it, either commercially or  
9 noncommercially. Secondly, this License preserves for the author and publisher a  
10 way to get credit for their work, while not being considered responsible for  
11 modifications made by others.

12           This License is a kind of "copyleft", which means that derivative works of the  
13 document must themselves be free in the same sense. It complements the GNU  
14 General Public License, which is a copyleft license designed for free software.

15           We have designed this License in order to use it for manuals for free software,  
16 because free software needs free documentation: a free program should come with  
17 manuals providing the same freedoms that the software does. But this License is not  
18 limited to software manuals; it can be used for any textual work, regardless of  
19 subject matter or whether it is published as a printed book. We recommend this  
20 License principally for works whose purpose is instruction or reference.

### B.2 APPLICABILITY AND DEFINITIONS

21           This License applies to any manual or other work that contains a notice placed by  
22 the copyright holder saying it can be distributed under the terms of this License. The  
23 "Document", below, refers to any such manual or work. Any member of the public is  
24 a licensee, and is addressed as "you".

25           A "Modified Version" of the Document means any work containing the Document or  
26 a portion of it, either copied verbatim, or with modifications and/or translated into  
27 another language.

28           A "Secondary Section" is a named appendix or a front-matter section of the  
29 Document that deals exclusively with the relationship of the publishers or authors of  
30 the Document to the Document's overall subject (or to related matters) and contains  
31 nothing that could fall directly within that overall subject. (For example, if the  
32 Document is in part a textbook of mathematics, a Secondary Section may not explain  
33 any mathematics.) The relationship could be a matter of historical connection with  
34 the subject or with related matters, or of legal, commercial, philosophical, ethical or  
35 political position regarding them.

36           The "Invariant Sections" are certain Secondary Sections whose titles are designated,  
37 as being those of Invariant Sections, in the notice that says that the Document is  
38 released under this License.

39           The "Cover Texts" are certain short passages of text that are listed, as Front-Cover  
40 Texts or Back-Cover Texts, in the notice that says that the Document is released  
41 under this License.



42 A "Transparent" copy of the Document means a machine-readable copy, represented  
43 in a format whose specification is available to the general public, whose contents can  
44 be viewed and edited directly and straightforwardly with generic text editors or (for  
45 images composed of pixels) generic paint programs or (for drawings) some widely  
46 available drawing editor, and that is suitable for input to text formatters or for  
47 automatic translation to a variety of formats suitable for input to text formatters. A  
48 copy made in an otherwise Transparent file format whose markup has been  
49 designed to thwart or discourage subsequent modification by readers is not  
50 Transparent. A copy that is not "Transparent" is called "Opaque".

51 Examples of suitable formats for Transparent copies include plain ASCII without  
52 markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly  
53 available DTD, and standard-conforming simple HTML designed for human  
54 modification. Opaque formats include PostScript, PDF, proprietary formats that can  
55 be read and edited only by proprietary word processors, SGML or XML for which  
56 the DTD and/or processing tools are not generally available, and the  
57 machine-generated HTML produced by some word processors for output purposes  
58 only.

59 The "Title Page" means, for a printed book, the title page itself, plus such following  
60 pages as are needed to hold, legibly, the material this License requires to appear in  
61 the title page. For works in formats which do not have any title page as such, "Title  
62 Page" means the text near the most prominent appearance of the work's title,  
63 preceding the beginning of the body of the text.

### B.3 VERBATIM COPYING

64 You may copy and distribute the Document in any medium, either commercially or  
65 noncommercially, provided that this License, the copyright notices, and the license  
66 notice saying this License applies to the Document are reproduced in all copies, and  
67 that you add no other conditions whatsoever to those of this License. You may not  
68 use technical measures to obstruct or control the reading or further copying of the  
69 copies you make or distribute. However, you may accept compensation in exchange  
70 for copies. If you distribute a large enough number of copies you must also follow  
71 the conditions in section 3.

72 You may also lend copies, under the same conditions stated above, and you may  
73 publicly display copies.

### B.4 COPYING IN QUANTITY

74 If you publish printed copies of the Document numbering more than 100, and the  
75 Document's license notice requires Cover Texts, you must enclose the copies in  
76 covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the  
77 front cover, and Back-Cover Texts on the back cover. Both covers must also clearly  
78 and legibly identify you as the publisher of these copies. The front cover must  
79 present the full title with all words of the title equally prominent and visible. You  
80 may add other material on the covers in addition. Copying with changes limited to  
81 the covers, as long as they preserve the title of the Document and satisfy these  
82 conditions, can be treated as verbatim copying in other respects.

83 If the required texts for either cover are too voluminous to fit legibly, you should put  
84 the first ones listed (as many as fit reasonably) on the actual cover, and continue the  
85 rest onto adjacent pages.

86 If you publish or distribute Opaque copies of the Document numbering more than  
87 100, you must either include a machine-readable Transparent copy along with each

88 Opaque copy, or state in or with each Opaque copy a publicly-accessible  
89 computer-network location containing a complete Transparent copy of the  
90 Document, free of added material, which the general network-using public has  
91 access to download anonymously at no charge using public-standard network  
92 protocols. If you use the latter option, you must take reasonably prudent steps, when  
93 you begin distribution of Opaque copies in quantity, to ensure that this Transparent  
94 copy will remain thus accessible at the stated location until at least one year after the  
95 last time you distribute an Opaque copy (directly or through your agents or  
96 retailers) of that edition to the public.

97 It is requested, but not required, that you contact the authors of the Document well  
98 before redistributing any large number of copies, to give them a chance to provide  
99 you with an updated version of the Document.

## B.5 MODIFICATIONS

100 You may copy and distribute a Modified Version of the Document under the  
101 conditions of sections 2 and 3 above, provided that you release the Modified Version  
102 under precisely this License, with the Modified Version filling the role of the  
103 Document, thus licensing distribution and modification of the Modified Version to  
104 whoever possesses a copy of it. In addition, you must do these things in the  
105 Modified Version:

- 106 A. Use in the Title Page (and on the covers, if any) a title distinct from that of the  
107 Document, and from those of previous versions (which should, if there were  
108 any, be listed in the History section of the Document). You may use the same  
109 title as a previous version if the original publisher of that version gives  
110 permission.
- 111 B. List on the Title Page, as authors, one or more persons or entities responsible  
112 for authorship of the modifications in the Modified Version, together with at  
113 least five of the principal authors of the Document (all of its principal authors,  
114 if it has less than five).
- 115 C. State on the Title page the name of the publisher of the Modified Version, as  
116 the publisher.
- 117 D. Preserve all the copyright notices of the Document.
- 118 E. Add an appropriate copyright notice for your modifications adjacent to the  
119 other copyright notices.
- 120 F. Include, immediately after the copyright notices, a license notice giving the  
121 public permission to use the Modified Version under the terms of this License,  
122 in the form shown in the Addendum below.
- 123 G. Preserve in that license notice the full lists of Invariant Sections and required  
124 Cover Texts given in the Document's license notice.
- 125 H. Include an unaltered copy of this License.
- 126 I. Preserve the section entitled "History", and its title, and add to it an item  
127 stating at least the title, year, new authors, and publisher of the Modified  
128 Version as given on the Title Page. If there is no section entitled "History" in  
129 the Document, create one stating the title, year, authors, and publisher of the  
130 Document as given on its Title Page, then add an item describing the Modified  
131 Version as stated in the previous sentence.
- 132 J. Preserve the network location, if any, given in the Document for public access  
133 to a Transparent copy of the Document, and likewise the network locations

- 134 given in the Document for previous versions it was based on. These may be  
135 placed in the "History" section. You may omit a network location for a work  
136 that was published at least four years before the Document itself, or if the  
137 original publisher of the version it refers to gives permission.
- 138 K. In any section entitled "Acknowledgements" or "Dedications", preserve the  
139 section's title, and preserve in the section all the substance and tone of each of  
140 the contributor acknowledgements and/or dedications given therein.
- 141 L. Preserve all the Invariant Sections of the Document, unaltered in their text and  
142 in their titles. Section numbers or the equivalent are not considered part of the  
143 section titles.
- 144 M. Delete any section entitled "Endorsements". Such a section may not be  
145 included in the Modified Version.
- 146 N. Do not retitle any existing section as "Endorsements" or to conflict in title with  
147 any Invariant Section.

148 If the Modified Version includes new front-matter sections or appendices that  
149 qualify as Secondary Sections and contain no material copied from the Document,  
150 you may at your option designate some or all of these sections as invariant. To do  
151 this, add their titles to the list of Invariant Sections in the Modified Version's license  
152 notice. These titles must be distinct from any other section titles.

153 You may add a section entitled "Endorsements", provided it contains nothing but  
154 endorsements of your Modified Version by various parties—for example, statements  
155 of peer review or that the text has been approved by an organization as the  
156 authoritative definition of a standard.

157 You may add a passage of up to five words as a Front-Cover Text, and a passage of  
158 up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the  
159 Modified Version. Only one passage of Front-Cover Text and one of Back-Cover  
160 Text may be added by (or through arrangements made by) any one entity. If the  
161 Document already includes a cover text for the same cover, previously added by you  
162 or by arrangement made by the same entity you are acting on behalf of, you may not  
163 add another; but you may replace the old one, on explicit permission from the  
164 previous publisher that added the old one.

165 The author(s) and publisher(s) of the Document do not by this License give  
166 permission to use their names for publicity for or to assert or imply endorsement of  
167 any Modified Version.

## B.6 COMBINING DOCUMENTS

168 You may combine the Document with other documents released under this License,  
169 under the terms defined in section 4 above for modified versions, provided that you  
170 include in the combination all of the Invariant Sections of all of the original  
171 documents, unmodified, and list them all as Invariant Sections of your combined  
172 work in its license notice.

173 The combined work need only contain one copy of this License, and multiple  
174 identical Invariant Sections may be replaced with a single copy. If there are multiple  
175 Invariant Sections with the same name but different contents, make the title of each  
176 such section unique by adding at the end of it, in parentheses, the name of the  
177 original author or publisher of that section if known, or else a unique number. Make  
178 the same adjustment to the section titles in the list of Invariant Sections in the license  
179 notice of the combined work.

180 In the combination, you must combine any sections entitled "History" in the various  
181 original documents, forming one section entitled "History"; likewise combine any  
182 sections entitled "Acknowledgements", and any sections entitled "Dedications". You  
183 must delete all sections entitled "Endorsements."

## **B.7 COLLECTIONS OF DOCUMENTS**

184 You may make a collection consisting of the Document and other documents  
185 released under this License, and replace the individual copies of this License in the  
186 various documents with a single copy that is included in the collection, provided  
187 that you follow the rules of this License for verbatim copying of each of the  
188 documents in all other respects.

189 You may extract a single document from such a collection, and distribute it  
190 individually under this License, provided you insert a copy of this License into the  
191 extracted document, and follow this License in all other respects regarding verbatim  
192 copying of that document.

## **B.8 AGGREGATION WITH INDEPENDENT WORKS**

193 A compilation of the Document or its derivatives with other separate and  
194 independent documents or works, in or on a volume of a storage or distribution  
195 medium, does not as a whole count as a Modified Version of the Document,  
196 provided no compilation copyright is claimed for the compilation. Such a  
197 compilation is called an "aggregate", and this License does not apply to the other  
198 self-contained works thus compiled with the Document, on account of their being  
199 thus compiled, if they are not themselves derivative works of the Document.

200 If the Cover Text requirement of section 3 is applicable to these copies of the  
201 Document, then if the Document is less than one quarter of the entire aggregate, the  
202 Document's Cover Texts may be placed on covers that surround only the Document  
203 within the aggregate. Otherwise they must appear on covers around the whole  
204 aggregate.

## **B.9 TRANSLATION**

205 Translation is considered a kind of modification, so you may distribute translations  
206 of the Document under the terms of section 4. Replacing Invariant Sections with  
207 translations requires special permission from their copyright holders, but you may  
208 include translations of some or all Invariant Sections in addition to the original  
209 versions of these Invariant Sections. You may include a translation of this License  
210 provided that you also include the original English version of this License. In case of  
211 a disagreement between the translation and the original English version of this  
212 License, the original English version will prevail.

## **B.10 TERMINATION**

213 You may not copy, modify, sublicense, or distribute the Document except as  
214 expressly provided for under this License. Any other attempt to copy, modify,  
215 sublicense or distribute the Document is void, and will automatically terminate your  
216 rights under this License. However, parties who have received copies, or rights,  
217 from you under this License will not have their licenses terminated so long as such  
218 parties remain in full compliance.

## B.11 FUTURE REVISIONS OF THIS LICENSE

219 The Free Software Foundation may publish new, revised versions of the GNU Free  
220 Documentation License from time to time. Such new versions will be similar in spirit  
221 to the present version, but may differ in detail to address new problems or concerns.  
222 See <http://www.gnu.org/copyleft/>.

223 Each version of the License is given a distinguishing version number. If the  
224 Document specifies that a particular numbered version of this License "or any later  
225 version" applies to it, you have the option of following the terms and conditions  
226 either of that specified version or of any later version that has been published (not as  
227 a draft) by the Free Software Foundation. If the Document does not specify a version  
228 number of this License, you may choose any version ever published (not as a draft)  
229 by the Free Software Foundation.

## B.12 How to use this License for your documents

230 To use this License in a document you have written, include a copy of the License in  
231 the document and put the following copyright and license notices just after the title  
232 page:

233 Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or  
234 modify this document under the terms of the GNU Free Documentation License, Version  
235 1.1 or any later version published by the Free Software Foundation; with the Invariant  
236 Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the  
237 Back-Cover Texts being LIST. A copy of the license is included in the section entitled  
238 "GNU Free Documentation License".

239 If you have no Invariant Sections, write "with no Invariant Sections" instead of  
240 saying which ones are invariant. If you have no Front-Cover Texts, write "no  
241 Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for  
242 Back-Cover Texts.

243 If your document contains nontrivial examples of program code, we recommend  
244 releasing these examples in parallel under your choice of free software license, such  
245 as the GNU General Public License, to permit their use in free software.