Linux Standard Base Core Specification for IA64 2.0.1

Linux Standard Base Core Specification for IA64 2.0.1

Specification Introduction

Specification Introduction

Table of Contents

Foreword i
Introduction ii
I. Introductory Elements 3
1. Scope 1
1.1. General 1
1.2. Module Specific Scope 1
2. Normative References 2
3. Requirements 5
3.1. Relevant Libraries 5
3.2. LSB Implementation Conformance 5
3.3. LSB Application Conformance 6
4. Definitions 7
5. Terminology 8
6. Documentation Conventions 9

List of Tables

2-1. Normative References 2
3-1. Standard Library Names..5

Foreword

This is version 2.0.1 of the Linux Standard Base Core Specification for IA64. An implementation of this version of the specification may not claim to be an implementation of the Linux Standard Base unless it has successfully completed the compliance process as defined by the Free Standards Group.

Introduction

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

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

I. Introductory Elements

Chapter 1. Scope

1.1. General

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

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

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

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

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

1.2. Module Specific Scope

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

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

Chapter 2. Normative References

The specifications listed below are referenced in whole or in part by the Linux Standard Base. In this specification, where only a particular section of one of these references is identified, then the normative reference is to that section alone, and the rest of the referenced document is informative.

Table 2-1. Normative References

Name	Title	URL
DWARF Debugging Information Format	DWARF Debugging Information Format, Revision 2.0.0 (July 27, 1993)	http://www.eagercon.com/dwarf/dwarf-2.0.0.pdf
Filesystem Hierarchy Standard	Filesystem Hierarchy Standard (FHS) 2.3	http://www.pathname.com/fhs/
IEEE Std 754-1985	IEEE Standard 754 for Binary Floating-Point Arithmetic	http://www.ieee.org/
Intel® Itanium ™ Processor-specific Application Binary Interface	Intel® Itanium ™ Processor-specific Application Binary Interface	http://refspecs.freestandards.org/elf/IA64-SysV-psABI.pdf
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 Information technology -- Portable Operating System Interface (POSIX) -- Part 4: Rationale	http://www.unix.org/version3/
Itanium ™ Architecture Software Developer's Manual Volume 1	Itanium ™ Architecture Software Developer's Manual Volume 1: Application Architecture	http://refspecs.freestandards.org/IA64-softdevman-vol1.pdf

Name	**Title**	**URL**
Itanium ™ Architecture Software Developer's Manual Volume 2	Itanium ™ Architecture Software Developer's Manual Volume 2: System Architecture	http://refspecs.freestandards.org/IA64-softdevman-vol2.pdf
Itanium ™ Architecture Software Developer's Manual Volume 3	Itanium ™ Architecture Software Developer's Manual Volume 3: Instruction Set Reference	http://refspecs.freestandards.org/IA64-softdevman-vol3.pdf
Itanium ™ Architecture Software Developer's Manual Volume 4	IA-64 Processor Reference: Intel® Itanium ™ Processor Reference Manual for Software Development	http://refspecs.freestandards.org/IA64-softdevman-vol4.pdf
Itanium ™ Software Conventions and Runtime Guide	Itanium ™ Software Conventions & Runtime Architecture Guide, September 2000	http://refspecs.freestandards.org/IA64conventions.pdf
Large File Support	Large File Support	http://www.UNIX-systems.org/version2/whatsnew/lfs20mar.html
Li18nux Globalization Specification	LI18NUX 2000 Globalization Specification, Version 1.0 with Amendment 4	http://www.li18nux.org/docs/html/LI18NUX-2000-amd4.htm
Linux Allocated Device Registry	LINUX ALLOCATED DEVICES	http://www.lanana.org/docs/device-list/devices.txt
PAM	Open Software Foundation, Request For Comments: 86.0 , October 1995, V. Samar & R.Schemers (SunSoft)	http://www.opengroup.org/tech/rfc/mirror-rfc/rfc86.0.txt
RFC 1321: The MD5 Message-Digest Algorithm	IETF RFC 1321: The MD5 Message-Digest Algorithm	http://www.ietf.org/rfc/rfc1321.txt
RFC 1833: Binding Protocols for ONC RPC Version 2	IETF RFC 1833: Binding Protocols for ONC RPC Version 2	http://www.ietf.org/rfc/rfc1833.txt
RFC 1951: DEFLATE Compressed Data Format Specification	IETF RFC 1951: DEFLATE Compressed Data Format Specification version 1.3	http://www.ietf.org/rfc/rfc1951.txt
RFC 1952: GZIP File Format Specification	IETF RFC 1952: GZIP file format specification version 4.3	http://www.ietf.org/rfc/rfc1952.txt
RFC 2440: OpenPGP Message Format	IETF RFC 2440: OpenPGP Message Format	http://www.ietf.org/rfc/rfc2440.txt
SUSv2	CAE Specification, January 1997, System Interfaces and Headers (XSH),Issue 5 (ISBN: 1-85912-181-0, C606)	http://www.opengroup.org/publications/catalog/un.htm
SUSv2 Command and Utilities	The Single UNIX®	http://www.opengroup.org/publicati

Name	Title	URL
	Specification(SUS) Version 2, Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)	ons/catalog/un.htm
SVID Issue 3	American Telephone and Telegraph Company, System V Interface Definition, Issue 3 ; 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	http://www.caldera.com/developers/devspecs/gabi41.pdf
System V ABI Update	System V Application Binary Interface - DRAFT - 17 December 2003	http://www.caldera.com/developers/gabi/2003-12-17/contents.html
this specification	Linux Standard Base	http://www.linuxbase.org/spec/
X/Open Curses	CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), plus Corrigendum U018	http://www.opengroup.org/publications/catalog/un.htm
zlib Manual	zlib 1.2 Manual	http://www.gzip.org/zlib/

Chapter 3. Requirements

3.1. Relevant Libraries

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

Table 3-1. Standard Library Names

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

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

3.2. LSB Implementation Conformance

A conforming implementation shall satisfy the following requirements:

- The implementation shall implement fully the architecture described in the hardware manual for the target processor architecture.
- The implementation shall be capable of executing compiled applications having the format and using the system interfaces described in this document.
- The implementation shall provide libraries containing the interfaces specified by this document, and shall provide a dynamic linking mechanism that allows these interfaces to be attached to applications at runtime. All the interfaces shall behave as specified in this document.
- The map of virtual memory provided by the implementation shall conform to the requirements of this document.
- The implementation's low-level behavior with respect to function call linkage, system traps, signals, and other such activities shall conform to the formats described in this document.

- The implementation shall provide all of the mandatory interfaces in their entirety.
- The implementation may provide one or more of the optional interfaces. Each optional interface that is provided shall be provided in its entirety. The product documentation shall state which optional interfaces are provided.
- The implementation shall provide all files and utilities specified as part of this document in the format defined here and in other referenced documents. All commands and utilities shall behave as required by this document. The implementation shall also provide all mandatory components of an application's runtime environment that are included or referenced in this document.
- The implementation, when provided with standard data formats and values at a named interface, shall provide the behavior defined for those values and data formats at that interface. However, a conforming implementation may consist of components which are separately packaged and/or sold. For example, a vendor of a conforming implementation might sell the hardware, operating system, and windowing system as separately packaged items.
- The implementation may provide additional interfaces with different names. It may also provide additional behavior corresponding to data values outside the standard ranges, for standard named interfaces.

3.3. LSB Application Conformance

A conforming application shall satisfy the following requirements:

- Its executable files are either shell scripts or object files in the format defined for the Object File Format system interface.
- Its object files participate in dynamic linking as defined in the Program Loading and Linking System interface.
- It employs only the instructions, traps, and other low-level facilities defined in the Low-Level System interface as being for use by applications.
- If it requires any optional interface defined in this document in order to be installed or to execute successfully, the requirement for that optional interface is stated in the application's documentation.
- It does not use any interface or data format that is not required to be provided by a conforming implementation, unless:
  - If such an interface or data format is supplied by another application through direct invocation of that application during execution, that application is in turn an LSB conforming application.
  - The use of that interface or data format, as well as its source, is identified in the documentation of the application.
- It shall not use any values for a named interface that are reserved for vendor extensions.

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

Chapter 4. Definitions

For the purposes of this document, the following definitions, as specified in the *ISO/IEC Directives, Part 2, 2001, 4th Edition*, apply:

can

: be able to; there is a possibility of; it is possible to

cannot

: be unable to; there is no possibilty of; it is not possible to

may

: is permitted; is allowed; is permissible

need not

: it is not required that; no...is required

shall

: is to; is required to; it is required that; has to; only...is permitted; it is necessary

shall not

: is not allowed [permitted] [acceptable] [permissible]; is required to be not; is required that...be not; is not to be

should

: it is recommended that; ought to

should not

: it is not recommended that; ought not to

Chapter 5. Terminology

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

archLSB

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

Binary Standard

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

gLSB

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

implementation-defined

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

Shell Script

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

Source Standard

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

undefined

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

unspecified

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

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

Chapter 6. Documentation Conventions

Throughout this document, the following typographic conventions are used:

`function()`

the name of a function

command

the name of a command or utility

`CONSTANT`

a constant value

`parameter`

a parameter

`variable`

a variable

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

name

the name of the interface

(symver)

An optional symbol version identifier, if required.

[*`refno`*]

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

For example,

forkpty(GLIBC_2.0) [1]

refers to the interface named `forkpty` with symbol version `GLIBC_2.0` that is defined in the first of the listed references below the table.

ELF Specification

ELF Specification

Table of Contents

I. Low Level System Information....16
1. Machine Interface....1
    1.1. Processor Architecture....1
    1.2. Data Representation....1
        1.2.1. Byte Ordering....2
        1.2.2. Fundamental Types....2
        1.2.3. Aggregates and Unions....3
        1.2.4. Bit Fields....4
2. Function Calling Sequence....6
    2.1. CPU Registers....6
    2.2. Floating Point Registers....6
    2.3. Stack Frame....6
    2.4. Arguments....6
        2.4.1. Integral/Pointer....6
        2.4.2. Floating Point....6
        2.4.3. Struct and Union Point....6
        2.4.4. Variable Arguments....6
    2.5. Return Values....6
        2.5.1. Void....7
        2.5.2. Integral/Pointer....7
        2.5.3. Floating Point....7
        2.5.4. Struct and Union....7
3. Operating System Interface....8
    3.1. Processor Execution Mode....8
    3.2. Exception Interface....8
        3.2.1. Hardware Exception Types....8
        3.2.2. Software Trap Types....8
        3.2.3. Debugging Support....8
        3.2.4. Process Startup....8
    3.3. Signal Delivery....8
        3.3.1. Signal Handler Interface....8
4. Process Initialization....9
    4.1. Special Registers....9
    4.2. Process Stack (on entry)....9
    4.3. Auxiliary Vector....9
    4.4. Environment....11
5. Coding Examples....12
    5.1. Code Model Overview/Architecture Constraints....12
    5.2. Position-Independent Function Prologue....12
    5.3. Data Objects....12
        5.3.1. Absolute Load & Store....12
        5.3.2. Position Relative Load & Store....12
    5.4. Function Calls....12

5.4.1. Absolute Direct Function Call ... 12
5.4.2. Absolute Indirect Function Call ... 13
5.4.3. Position-Independent Direct Function Call ... 13
5.4.4. Position-Independent Indirect Function Call ... 13
5.5. Branching ... 13
5.5.1. Branch Instruction ... 13
5.5.2. Absolute switch() code ... 13
5.5.3. Position-Independent switch() code ... 13
6. C Stack Frame ... 14
6.1. Variable Argument List ... 14
6.2. Dynamic Allocation of Stack Space ... 14
7. Debug Information ... 15
II. Object Format ... 16
8. ELF Header ... 17
8.1. Machine Information ... 17
8.1.1. File Class ... 17
8.1.2. Data Encoding ... 17
8.1.3. OS Identification ... 17
8.1.4. Processor Identification ... 17
8.1.5. Processor Specific Flags ... 17
9. Sections ... 18
9.1. Special Sections ... 18
9.2. Linux Special Sections ... 19
9.3. Section Types ... 20
9.4. Section Attribute Flags ... 20
9.5. Special Section Types ... 20
10. Symbol Table ... 21
11. Relocation ... 22
11.1. Relocation Types ... 22
III. Program Loading and Dynamic Linking ... 23
12. Program Header ... 24
12.1. Types ... 24
12.2. Flags ... 24
13. Program Loading ... 25
14. Dynamic Linking ... 26
14.1. Dynamic Entries ... 26
14.1.1. ELF Dynamic Entries ... 26
14.1.2. Additional Dynamic Entries ... 26
14.2. Global Offset Table ... 26
14.3. Shared Object Dependencies ... 26
14.4. Function Addresses ... 26
14.5. Procedure Linkage Table ... 26
14.6. Initialization and Termination Functions ... 26

List of Tables

1-1. Scalar Types 2
8-1. Additional Processor-Specific Flags 17
9-1. ELF Special Sections 18
9-2. Additional Special Sections 19

List of Figures

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

I. Low Level System Information

Chapter 1. Machine Interface

1.1. Processor Architecture

The Architecture is specified by the following documents

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

Only the features of the processor instruction set may be assumed to be present. An application is responsible for determining if any additional instruction set features are available before using those additional features. If a feature is not present, then the application may not use it.

Only instructions which do not require elevated privileges may be used.

Applications may not make system calls directly. The interfaces in the C library must be used instead.

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

Applications conforming to this specification must provide feedback to the user if a feature that is required for correct execution of the application is not present. Applications conforming to this specification should attempt to execute in a diminished capacity if a required feature is not present.

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

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

1.2. Data Representation

See Itanium ™ Software Conventions and Runtime Guide Chapter 4.

Within this specification, the term `byte` refers to an 8-bit object, the term `halfword` refers to a 16-bit object, the term `word` refers to a 32-bit object, the term `doubleword` refers to a 64-bit object, and the term `quadword` refers to a 128-bit object. Although the architecture also supports 120-bit addressable objects, this specification does not require LSB-conforming implementations to provide support for these objects.

1.2.1. Byte Ordering

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

1.2.2. Fundamental Types

Table 2-1 describes how fundemental C language data types shall be represented:

Table 1-1. Scalar Types

Type	C	`sizeof`	Alignment (bytes)	Notes
Integral	char	1	1	
	signed char			
	unsigned char			
	short	2	2	
	signed short			
	unsigned short			
	int	4	4	
	signed int			
	unsigned int			
	long	8	8	
	signed long			
	unsigned long			
	long long	8	8	See Note Below
	signed long long			
	unsigned long long			
Pointer	*`any-type`* *	8	8	
	`any-type (*)()`			
Floating-Point	float	4	4	
	double	8	8	
	long double	16	16	

> Support for the `long long` data type is dependent on support for ISO9899:1999 C language. This standard is not required for LSB-conformance, but this data type is important when developing applications for the architecture. The GNU Compiler Collection (gcc) includes support for `long long` of ISO9899:1999.

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

1.2.3. Aggregates and Unions

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.

- An entire structure or union object shall be aligned on the same boundary as its most strictly aligned member.
- Each member shall be assigned to the lowest available offset with the appropriate alignment. This may require *internal padding*, depending on the previous member.
- 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.

A conforming application shall not read padding.

Figure 1-1. Structure Smaller Than A Word

```
struct {
    char c;
}
```

Byte aligned, `sizeof` is 1

Offset	Byte 0
0	c^0

Figure 1-2. No Padding

```
struct {
    char  c;
    char  d;
    short s;
    int   i;
    long  l;
}
```

Doubleword Aligned, `sizeof` is 16

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0	s^2		d^1	c^0
4	i^0			
8	l^0			
12				

Figure 1-3. Internal and Tail Padding

```
struct {
    char  c;
    long  l;
    int   i;
    short s;
}
```

<table>
<tr><td colspan="5">Doubleword Aligned, <code>sizeof</code> is 24</td></tr>
<tr><th>Offset</th><th>Byte 3</th><th>Byte 2</th><th>Byte 1</th><th>Byte 0</th></tr>
<tr><td>0</td><td colspan="3">pad^1</td><td>c^0</td></tr>
<tr><td>4</td><td colspan="4">pad^1</td></tr>
<tr><td>8</td><td colspan="4" rowspan="2">l^0</td></tr>
<tr><td>12</td></tr>
<tr><td>16</td><td colspan="4">i^0</td></tr>
<tr><td>20</td><td colspan="2">pad^2</td><td colspan="2">s^0</td></tr>
</table>

1.2.4. Bit Fields

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

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:

- 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.

Figure 1-4. Bit-Field Ranges

Bit-field Type	Width	Range

`signed char`
`char`
`unsigned char`
	1 to 8	-2^{-1} to $2^{-1}-1$
		`0 to 2-1`
		`0 to 2-1`

`signed short` `short`	1 to 16	-2^{-1} to $2^{-1}-1$ `0 to 2-1`

Bit-field Type	Width	Range
`unsigned short`		`0 to 2-1`

`signed int`
`int`
`unsigned int`
	1 to 32	-2^{-1} to $2^{-1}-1$
		`0 to 2-1`
		`0 to 2-1`

`signed long`
`long`
`unsigned long`
	1 to 64	-2^{-1} to $2^{-1}-1$
		`0 to 2-1`
		`0 to 2-1`

69

Chapter 2. Function Calling Sequence

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.

2.1. CPU Registers

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

2.2. Floating Point Registers

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

2.3. Stack Frame

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

2.4. Arguments

The procedure argument passing mechanism is as described in the Itanium ™ Software Conventions and Runtime Guide Chapter 8.5.

2.4.1. Integral/Pointer

See Itanium ™ Software Conventions and Runtime Guide Chapter 8.5.

2.4.2. Floating Point

See Itanium ™ Software Conventions and Runtime Guide Chapter 8.5.

2.4.3. Struct and Union Point

See Itanium ™ Software Conventions and Runtime Guide Chapter 8.5.

2.4.4. Variable Arguments

See Itanium ™ Software Conventions and Runtime Guide Chapter 8.5.4.

2.5. Return Values

See Itanium ™ Software Conventions and Runtime Guide Chapter 8.6.

2.5.1. Void

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

2.5.2. Integral/Pointer

See Itanium ™ Software Conventions and Runtime Guide Chapter 8.6.

2.5.3. Floating Point

See Itanium ™ Software Conventions and Runtime Guide Chapter 8.6.

2.5.4. Struct and Union

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

Chapter 3. Operating System Interface

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

3.1. Processor Execution Mode

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

3.2. Exception Interface

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

3.2.1. Hardware Exception Types

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

3.2.2. Software Trap Types

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

3.2.3. Debugging Support

See Intel® Itanium ™ Processor-specific Application Binary Interface, section 3.3.4.

3.2.4. Process Startup

See Intel® Itanium ™ Processor-specific Application Binary Interface, section 3.3.5.

3.3. Signal Delivery

See Intel® Itanium ™ Processor-specific Application Binary Interface, section 3.3.2.

3.3.1. Signal Handler Interface

See Intel® Itanium ™ Processor-specific Application Binary Interface, section 3.3.3.

Chapter 4. Process Initialization

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

4.1. Special Registers

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

4.2. Process Stack (on entry)

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

4.3. Auxiliary Vector

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

```
typedef struct
{
  long int a_type;              /* Entry type */
  union
    {
      long int a_val;           /* Integer value */
      void *a_ptr;              /* Pointer value */
      void (*a_fcn) (void);     /* Function pointer value */
    } a_un;
} auxv_t;
```

The application shall interpret the `a_un` value according to the `a_type`. Other auxiliary vector types are reserved.

The `a_type` field shall contain one of the following values:

AT_NULL

The last entry in the array has type `AT_NULL`. The value in `a_un` is undefined.

AT_IGNORE

The value in `a_un` is undefined, and should be ignored.

AT_EXECFD
    File descriptor of program
AT_PHDR
    Program headers for program
AT_PHENT
    Size of program header entry
AT_PHNUM
    Number of program headers
AT_PAGESZ
    System page size
AT_BASE
    Base address of interpreter
AT_FLAGS
    Flags
AT_ENTRY
    Entry point of program
AT_NOTELF
    Program is not ELF
AT_UID
    Real uid
AT_EUID
    Effective uid
AT_GID
    Real gid
AT_EGID
    Effective gid
AT_CLKTCK
    Frequency of times()
AT_PLATFORM
    String identifying platform.

AT_HWCAP

Machine dependent hints about processor capabilities.

AT_FPUCW

Used FPU control word

AT_DCACHEBSIZE

Data cache block size

AT_ICACHEBSIZE

Instruction cache block size

AT_UCACHEBSIZE

Unified cache block size

The auxiliary vector is intended for passing information from the operating system to the program interpreter.

4.4. Environment

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

Chapter 5. Coding Examples

LSB-conforming applications may implement fundamental operations using the Coding Examples as shown below.

Sample code sequences and coding conventions can be found in Itanium ™ Software Conventions and Runtime Guide, Chapter 9.

5.1. Code Model Overview/Architecture Constraints

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

5.2. Position-Independent Function Prologue

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

5.3. Data Objects

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

5.3.1. Absolute Load & Store

Conforming applications shall not use absolute addressing.

5.3.2. Position Relative Load & Store

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

5.4. Function Calls

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

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

5.4.1. Absolute Direct Function Call

Conforming applications shall not use absolute addressing.

5.4.2. Absolute Indirect Function Call

Conforming applications shall not use absolute addressing.

5.4.3. Position-Independent Direct Function Call

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

5.4.4. Position-Independent Indirect Function Call

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

5.5. Branching

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

5.5.1. Branch Instruction

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

5.5.2. Absolute switch() code

Conforming applications shall not use absolute addressing.

5.5.3. Position-Independent switch() code

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

Chapter 6. C Stack Frame

6.1. Variable Argument List

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

6.2. Dynamic Allocation of Stack Space

2 The C library `alloca` function should be used to dynamically allocate stack space.

Chapter 7. Debug Information

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

II. Object Format

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

Chapter 8. ELF Header

8.1. Machine Information

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

8.1.1. File Class

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

8.1.2. Data Encoding

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

8.1.3. OS Identification

The OS Identification field `e_ident[EI_OSABI]` shall contain the value ELFOSABI_LINUX.

8.1.4. Processor Identification

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

8.1.5. Processor Specific Flags

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

The following additional processor-specific flags are defined:

Table 8-1. Additional Processor-Specific Flags

Name	Value
EF_IA_64_LINUX_EXECUTABLE_STACK	0x00000001

EF_IA_64_LINUX_EXECUTABLE_STACK

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

Chapter 9. Sections

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

9.1. Special Sections

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

Table 9-1. 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
.sdata	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE+SHF_IA_64_SHORT
.sdata1	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE+SHF_IA_64_SHORT

.got

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

.IA_64.archext

This section holds product-specific extension bits. The link editor will perform a logical "or" of the extension bits of each object when creating an executable so that it creates only a single .IA_64.archext section in the executable.

.IA_64.pltoff

This section holds local function descriptor entries.

.IA_64.unwind

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

.IA_64.unwind_info

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

.plt

This section holds the Procedure Linkage Table.

.sbss

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

.sdata

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

.sdata1

See .sdata.

9.2. Linux Special Sections

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

Table 9-2. Additional Special Sections

Name	Type	Attributes
.opd	SHT_PROGBITS	SHF_ALLOC
.rela.dyn	SHT_RELA	SHF_ALLOC
.rela.IA_64.pltoff	SHT_RELA	SHF_ALLOC

.opd

This section holds function descriptors

.rela.dyn

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

.rela.IA_64.pltoff

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

9.3. Section Types

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

9.4. Section Attribute Flags

See Intel® Itanium ™ Processor-specific Application Binary Interface, Chapter 4.2.2.

9.5. Special Section Types

See Intel® Itanium ™ Processor-specific Application Binary Interface, Chapter 4.2.3.

Chapter 10. Symbol Table

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

Need to add something here about st_info and st_other ...

Chapter 11. Relocation

LSB-conforming applications shall use Relocations as defined in Intel® Itanium ™ Processor-specific Application Binary Interface, Chapter 4.3.

11.1. Relocation Types

See Intel® Itanium ™ Processor-specific Application Binary Interface, Chapter 4.3.

III. Program Loading and Dynamic Linking

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

Chapter 12. Program Header

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

12.1. Types

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

12.2. Flags

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

Chapter 13. Program Loading

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

Chapter 14. Dynamic Linking

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

14.1. Dynamic Entries

14.1.1. ELF Dynamic Entries

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

DT_PLTGOT

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

14.1.2. Additional Dynamic Entries

The following dynamic entries are defined here.

DT_RELACOUNT

The number of relative relocations in .rela.dyn

14.2. Global Offset Table

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

14.3. Shared Object Dependencies

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

14.4. Function Addresses

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

14.5. Procedure Linkage Table

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

14.6. Initialization and Termination Functions

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

Linux Standard Base Specification

Linux Standard Base Specification

Table of Contents

I. Base Libraries **33**
1. Libraries 1
1.1. Program Interpreter/Dynamic Linker 1
1.2. Interfaces for libc 1
1.2.1. RPC 1
1.2.1.1. Interfaces for RPC 1
1.2.2. System Calls 2
1.2.2.1. Interfaces for System Calls 2
1.2.3. Standard I/O 4
1.2.3.1. Interfaces for Standard I/O 4
1.2.4. Signal Handling 5
1.2.4.1. Interfaces for Signal Handling 5
1.2.5. Localization Functions 6
1.2.5.1. Interfaces for Localization Functions 6
1.2.6. Socket Interface 7
1.2.6.1. Interfaces for Socket Interface 7
1.2.7. Wide Characters 8
1.2.7.1. Interfaces for Wide Characters 8
1.2.8. String Functions 9
1.2.8.1. Interfaces for String Functions 9
1.2.9. IPC Functions 10
1.2.9.1. Interfaces for IPC Functions 10
1.2.10. Regular Expressions 11
1.2.10.1. Interfaces for Regular Expressions 11
1.2.11. Character Type Functions 11
1.2.11.1. Interfaces for Character Type Functions 11
1.2.12. Time Manipulation 12
1.2.12.1. Interfaces for Time Manipulation 12
1.2.13. Terminal Interface Functions 13
1.2.13.1. Interfaces for Terminal Interface Functions 13
1.2.14. System Database Interface 14
1.2.14.1. Interfaces for System Database Interface 14
1.2.15. Language Support 14
1.2.15.1. Interfaces for Language Support 14
1.2.16. Large File Support 15
1.2.16.1. Interfaces for Large File Support 15
1.2.17. Standard Library 15
1.2.17.1. Interfaces for Standard Library 15
1.3. Data Definitions for libc 17
1.3.1. errno.h 18
1.3.2. inttypes.h 18
1.3.3. limits.h 18
1.3.4. setjmp.h 18

1.3.5. signal.h 18
1.3.6. stddef.h 19
1.3.7. sys/ioctl.h 19
1.3.8. sys/ipc.h 20
1.3.9. sys/mman.h 20
1.3.10. sys/msg.h 20
1.3.11. sys/sem.h 20
1.3.12. sys/shm.h 21
1.3.13. sys/socket.h 21
1.3.14. sys/stat.h 21
1.3.15. sys/statvfs.h 22
1.3.16. sys/types.h 23
1.3.17. termios.h 23
1.3.18. ucontext.h 24
1.3.19. unistd.h 24
1.3.20. utmp.h 25
1.3.21. utmpx.h 25
1.4. Interfaces for libm 25
1.4.1. Math 26
1.4.1.1. Interfaces for Math 26
1.5. Interfaces for libpthread 29
1.5.1. Realtime Threads 30
1.5.1.1. Interfaces for Realtime Threads 30
1.5.2. Advanced Realtime Threads 30
1.5.2.1. Interfaces for Advanced Realtime Threads 30
1.5.3. Posix Threads 30
1.5.3.1. Interfaces for Posix Threads 30
1.6. Interfaces for libgcc_s 31
1.6.1. Unwind Library 32
1.6.1.1. Interfaces for Unwind Library 32
1.7. Interface Definitions for libgcc_s 32
_Unwind_DeleteException 33
_Unwind_ForcedUnwind 34
_Unwind_GetGR 35
_Unwind_GetIP 35
_Unwind_GetLanguageSpecificData 35
_Unwind_GetRegionStart 36
_Unwind_RaiseException 37
_Unwind_Resume 38
_Unwind_SetGR 38
_Unwind_SetIP 38
1.8. Interfaces for libdl 38
1.8.1. Dynamic Loader 39
1.8.1.1. Interfaces for Dynamic Loader 39
1.9. Interfaces for libcrypt 39
1.9.1. Encryption 39
1.9.1.1. Interfaces for Encryption 39

II. Utility Libraries 41

2. Libraries 42

2.1. Interfaces for libz 42

2.1.1. Compression Library 42

2.1.1.1. Interfaces for Compression Library 42

2.2. Interfaces for libncurses 42

2.2.1. Curses 42

2.2.1.1. Interfaces for Curses 42

2.3. Interfaces for libutil 42

2.3.1. Utility Functions 43

2.3.1.1. Interfaces for Utility Functions 43

A. Alphabetical Listing of Interfaces 44

A.1. libgcc_s 44

List of Tables

1-1. libc Definition........1
1-2. libc - RPC Function Interfaces........1
1-3. libc - System Calls Function Interfaces........2
1-4. libc - Standard I/O Function Interfaces........4
1-5. libc - Standard I/O Data Interfaces........5
1-6. libc - Signal Handling Function Interfaces........5
1-7. libc - Signal Handling Data Interfaces........6
1-8. libc - Localization Functions Function Interfaces........6
1-9. libc - Localization Functions Data Interfaces........7
1-10. libc - Socket Interface Function Interfaces........7
1-11. libc - Socket Interface Deprecated Function Interfaces........8
1-12. libc - Wide Characters Function Interfaces........8
1-13. libc - String Functions Function Interfaces........9
1-14. libc - IPC Functions Function Interfaces........10
1-15. libc - Regular Expressions Function Interfaces........11
1-16. libc - Regular Expressions Deprecated Function Interfaces........11
1-17. libc - Regular Expressions Deprecated Data Interfaces........11
1-18. libc - Character Type Functions Function Interfaces........12
1-19. libc - Time Manipulation Function Interfaces........12
1-20. libc - Time Manipulation Deprecated Function Interfaces........13
1-21. libc - Time Manipulation Data Interfaces........13
1-22. libc - Terminal Interface Functions Function Interfaces........13
1-23. libc - System Database Interface Function Interfaces........14
1-24. libc - Language Support Function Interfaces........14
1-25. libc - Large File Support Function Interfaces........15
1-26. libc - Standard Library Function Interfaces........15
1-27. libc - Standard Library Data Interfaces........17
1-28. libm Definition........26
1-29. libm - Math Function Interfaces........26
1-30. libm - Math Data Interfaces........29
1-31. libpthread Definition........29
1-32. libpthread - Posix Threads Function Interfaces........30
1-33. libgcc_s Definition........32
1-34. libgcc_s - Unwind Library Function Interfaces........32
1-35. libdl Definition........39
1-36. libdl - Dynamic Loader Function Interfaces........39
1-37. libcrypt Definition........39
1-38. libcrypt - Encryption Function Interfaces........39
2-1. libz Definition........42
2-2. libncurses Definition........42
2-3. libutil Definition........42
2-4. libutil - Utility Functions Function Interfaces........43
A-1. libgcc_s Function Interfaces........44

I. Base Libraries

Chapter 1. Libraries

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

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

1.1. Program Interpreter/Dynamic Linker

The LSB specifies the Program Interpreter to be /lib/ld-lsb-ia64.so.2.

1.2. Interfaces for libc

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

Table 1-1. libc Definition

Library:	libc
SONAME:	libc.so.6.1

The behavior of the interfaces in this library is specified by the following specifications:

Large File Support
this specification
SUSv2
ISO POSIX (2003)
SVID Issue 3
SVID Issue 4

1.2.1. RPC

1.2.1.1. Interfaces for RPC

An LSB conforming implementation shall provide the architecture specific functions for RPC specified in Table 1-2, with the full functionality as described in the referenced underlying specification.

Table 1-2. libc - RPC Function Interfaces

authnone_create(GLIBC_2.2) [1]	pmap_unset(GLIBC_2.2) [2]	svcerr_weakauth(GLIBC_2.2) [3]	xdr_float(GLIBC_2.2) [3]	xdr_u_char(GLIBC_2.2) [3]
clnt_create(GLIBC_2.2) [1]	setdomainname(GLIBC_2.2) [2]	svctcp_create(GLIBC_2.2) [2]	xdr_free(GLIBC_2.2) [3]	xdr_u_int(GLIBC_2.2) [2]
clnt_pcreateerror(GLIBC_2.2) [1]	svc_getreqset(GLIBC_2.2) [3]	svcudp_create(GLIBC_2.2) [2]	xdr_int(GLIBC_2.2) [3]	xdr_u_long(GLIBC_2.2) [3]

clnt_perrno(GLIBC_2.2) [1]	svc_register(GLIBC_2.2) [2]	xdr_accepted_reply(GLIBC_2.2) [3]	xdr_long(GLIBC_2.2) [3]	xdr_u_short(GLIBC_2.2) [3]
clnt_perror(GLIBC_2.2) [1]	svc_run(GLIBC_2.2) [2]	xdr_array(GLIBC_2.2) [3]	xdr_opaque(GLIBC_2.2) [3]	xdr_union(GLIBC_2.2) [3]
clnt_spcreateerror(GLIBC_2.2) [1]	svc_sendreply(GLIBC_2.2) [2]	xdr_bool(GLIBC_2.2) [3]	xdr_opaque_auth(GLIBC_2.2) [3]	xdr_vector(GLIBC_2.2) [3]
clnt_sperrno(GLIBC_2.2) [1]	svcerr_auth(GLIBC_2.2) [3]	xdr_bytes(GLIBC_2.2) [3]	xdr_pointer(GLIBC_2.2) [3]	xdr_void(GLIBC_2.2) [3]
clnt_sperror(GLIBC_2.2) [1]	svcerr_decode(GLIBC_2.2) [3]	xdr_callhdr(GLIBC_2.2) [3]	xdr_reference(GLIBC_2.2) [3]	xdr_wrapstring(GLIBC_2.2) [3]
getdomainname(GLIBC_2.2) [2]	svcerr_noproc(GLIBC_2.2) [3]	xdr_callmsg(GLIBC_2.2) [3]	xdr_rejected_reply(GLIBC_2.2) [3]	xdrmem_create(GLIBC_2.2) [3]
key_decryptsession(GLIBC_2.2) [3]	svcerr_noprog(GLIBC_2.2) [3]	xdr_char(GLIBC_2.2) [3]	xdr_replymsg(GLIBC_2.2) [3]	xdrrec_create(GLIBC_2.2) [3]
pmap_getport(GLIBC_2.2) [2]	svcerr_progvers(GLIBC_2.2) [3]	xdr_double(GLIBC_2.2) [3]	xdr_short(GLIBC_2.2) [3]	xdrrec_eof(GLIBC_2.2) [3]
pmap_set(GLIBC_2.2) [2]	svcerr_systemerr(GLIBC_2.2) [3]	xdr_enum(GLIBC_2.2) [3]	xdr_string(GLIBC_2.2) [3]	

Referenced Specification(s)

[1]. SVID Issue 4

[2]. this specification

[3]. SVID Issue 3

1.2.2. System Calls

1.2.2.1. Interfaces for System Calls

An LSB conforming implementation shall provide the architecture specific functions for System Calls specified in Table 1-3, with the full functionality as described in the referenced underlying specification.

Table 1-3. libc - System Calls Function Interfaces

__fxstat(GLIBC_2.2) [1]	fchmod(GLIBC_2.2) [2]	getwd(GLIBC_2.2) [2]	read(GLIBC_2.2) [2]	setrlimit(GLIBC_2.2) [2]
__getpgid(GLIBC_2.2) [1]	fchown(GLIBC_2.2) [2]	initgroups(GLIBC_2.2) [1]	readdir(GLIBC_2.2) [2]	setrlimit64(GLIBC_2.2) [3]
__lxstat(GLIBC_2.2) [1]	fcntl(GLIBC_2.2) [1]	ioctl(GLIBC_2.2) [1]	readdir_r(GLIBC_2.2) [2]	setsid(GLIBC_2.2) [2]
__xmknod(GLIBC_2.2) [1]	fdatasync(GLIBC_2.2) [2]	kill(GLIBC_2.2) [1]	readlink(GLIBC_2.2) [2]	setuid(GLIBC_2.2) [2]

__xstat(GLIBC_2.2) [1]	flock(GLIBC_2.2) [1]	killpg(GLIBC_2.2) [2]	readv(GLIBC_2.2) [2]	sleep(GLIBC_2.2) [2]
access(GLIBC_2.2) [2]	fork(GLIBC_2.2) [2]	lchown(GLIBC_2.2) [2]	rename(GLIBC_2.2) [2]	statvfs(GLIBC_2.2) [2]
acct(GLIBC_2.2) [1]	fstatvfs(GLIBC_2.2) [2]	link(GLIBC_2.2) [2]	rmdir(GLIBC_2.2) [2]	stime(GLIBC_2.2) [1]
alarm(GLIBC_2.2) [2]	fsync(GLIBC_2.2) [2]	lockf(GLIBC_2.2) [2]	sbrk(GLIBC_2.2) [4]	symlink(GLIBC_2.2) [2]
brk(GLIBC_2.2) [4]	ftime(GLIBC_2.2) [2]	lseek(GLIBC_2.2) [2]	sched_get_priority_max(GLIBC_2.2) [2]	sync(GLIBC_2.2) [2]
chdir(GLIBC_2.2) [2]	ftruncate(GLIBC_2.2) [2]	mkdir(GLIBC_2.2) [2]	sched_get_priority_min(GLIBC_2.2) [2]	sysconf(GLIBC_2.2) [2]
chmod(GLIBC_2.2) [2]	getcontext(GLIBC_2.2) [2]	mkfifo(GLIBC_2.2) [2]	sched_getparam(GLIBC_2.2) [2]	time(GLIBC_2.2) [2]
chown(GLIBC_2.2) [2]	getegid(GLIBC_2.2) [2]	mlock(GLIBC_2.2) [2]	sched_getscheduler(GLIBC_2.2) [2]	times(GLIBC_2.2) [2]
chroot(GLIBC_2.2) [4]	geteuid(GLIBC_2.2) [2]	mlockall(GLIBC_2.2) [2]	sched_rr_get_interval(GLIBC_2.2) [2]	truncate(GLIBC_2.2) [2]
clock(GLIBC_2.2) [2]	getgid(GLIBC_2.2) [2]	mmap(GLIBC_2.2) [2]	sched_setparam(GLIBC_2.2) [2]	ulimit(GLIBC_2.2) [2]
close(GLIBC_2.2) [2]	getgroups(GLIBC_2.2) [2]	mprotect(GLIBC_2.2) [2]	sched_setscheduler(GLIBC_2.2) [2]	umask(GLIBC_2.2) [2]
closedir(GLIBC_2.2) [2]	getitimer(GLIBC_2.2) [2]	msync(GLIBC_2.2) [2]	sched_yield(GLIBC_2.2) [2]	uname(GLIBC_2.2) [2]
creat(GLIBC_2.2) [1]	getloadavg(GLIBC_2.2) [1]	munlock(GLIBC_2.2) [2]	select(GLIBC_2.2) [2]	unlink(GLIBC_2.2) [1]
dup(GLIBC_2.2) [2]	getpagesize(GLIBC_2.2) [4]	munlockall(GLIBC_2.2) [2]	setcontext(GLIBC_2.2) [2]	utime(GLIBC_2.2) [2]
dup2(GLIBC_2.2) [2]	getpgid(GLIBC_2.2) [2]	munmap(GLIBC_2.2) [2]	setegid(GLIBC_2.2) [2]	utimes(GLIBC_2.2) [2]
execl(GLIBC_2.2) [2]	getpgrp(GLIBC_2.2) [2]	nanosleep(GLIBC_2.2) [2]	seteuid(GLIBC_2.2) [2]	vfork(GLIBC_2.2) [2]
execle(GLIBC_2.2) [2]	getpid(GLIBC_2.2) [2]	nice(GLIBC_2.2) [2]	setgid(GLIBC_2.2) [2]	wait(GLIBC_2.2) [2]
execlp(GLIBC_2.2) [2]	getppid(GLIBC_2.2) [2]	open(GLIBC_2.2) [1]	setitimer(GLIBC_2.2) [2]	wait3(GLIBC_2.2) [1]

execv(GLIBC_2.2) [2]	getpriority(GLIBC_2.2) [2]	opendir(GLIBC_2.2) [2]	setpgid(GLIBC_2.2) [2]	wait4(GLIBC_2.2) [1]
execve(GLIBC_2.2) [2]	getrlimit(GLIBC_2.2) [2]	pathconf(GLIBC_2.2) [2]	setpgrp(GLIBC_2.2) [2]	waitpid(GLIBC_2.2) [1]
execvp(GLIBC_2.2) [2]	getrusage(GLIBC_2.2) [2]	pause(GLIBC_2.2) [2]	setpriority(GLIBC_2.2) [2]	write(GLIBC_2.2) [2]
exit(GLIBC_2.2) [2]	getsid(GLIBC_2.2) [2]	pipe(GLIBC_2.2) [2]	setregid(GLIBC_2.2) [2]	writev(GLIBC_2.2) [2]
fchdir(GLIBC_2.2) [2]	getuid(GLIBC_2.2) [2]	poll(GLIBC_2.2) [2]	setreuid(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

[3]. Large File Support

[4]. SUSv2

1.2.3. Standard I/O

1.2.3.1. Interfaces for Standard I/O

An LSB conforming implementation shall provide the architecture specific functions for Standard I/O specified in Table 1-4, with the full functionality as described in the referenced underlying specification.

Table 1-4. libc - Standard I/O Function Interfaces

_IO_feof(GLIBC_2.2) [1]	fgetpos(GLIBC_2.2) [2]	fsetpos(GLIBC_2.2) [2]	putchar(GLIBC_2.2) [2]	sscanf(GLIBC_2.2) [2]
_IO_getc(GLIBC_2.2) [1]	fgets(GLIBC_2.2) [2]	ftell(GLIBC_2.2) [2]	putchar_unlocked(GLIBC_2.2) [2]	telldir(GLIBC_2.2) [2]
_IO_putc(GLIBC_2.2) [1]	fgetwc_unlocked(GLIBC_2.2) [1]	ftello(GLIBC_2.2) [2]	puts(GLIBC_2.2) [2]	tempnam(GLIBC_2.2) [2]
_IO_puts(GLIBC_2.2) [1]	fileno(GLIBC_2.2) [2]	fwrite(GLIBC_2.2) [2]	putw(GLIBC_2.2) [3]	ungetc(GLIBC_2.2) [2]
asprintf(GLIBC_2.2) [1]	flockfile(GLIBC_2.2) [2]	getc(GLIBC_2.2) [2]	remove(GLIBC_2.2) [2]	vasprintf(GLIBC_2.2) [1]
clearerr(GLIBC_2.2) [2]	fopen(GLIBC_2.2) [1]	getc_unlocked(GLIBC_2.2) [2]	rewind(GLIBC_2.2) [2]	vdprintf(GLIBC_2.2) [1]
ctermid(GLIBC_2.2) [2]	fprintf(GLIBC_2.2) [2]	getchar(GLIBC_2.2) [2]	rewinddir(GLIBC_2.2) [2]	vfprintf(GLIBC_2.2) [2]

fclose(GLIBC_2.2) [2]	fputc(GLIBC_2.2) [2]	getchar_unlocked(GLIBC_2.2) [2]	scanf(GLIBC_2.2) [2]	vprintf(GLIBC_2.2) [2]
fdopen(GLIBC_2.2) [2]	fputs(GLIBC_2.2) [2]	getw(GLIBC_2.2) [3]	seekdir(GLIBC_2.2) [2]	vsnprintf(GLIBC_2.2) [2]
feof(GLIBC_2.2) [2]	fread(GLIBC_2.2) [2]	pclose(GLIBC_2.2) [2]	setbuf(GLIBC_2.2) [2]	vsprintf(GLIBC_2.2) [2]
ferror(GLIBC_2.2) [2]	freopen(GLIBC_2.2) [1]	popen(GLIBC_2.2) [2]	setbuffer(GLIBC_2.2) [1]	
fflush(GLIBC_2.2) [2]	fscanf(GLIBC_2.2) [2]	printf(GLIBC_2.2) [2]	setvbuf(GLIBC_2.2) [2]	
fflush_unlocked(GLIBC_2.2) [1]	fseek(GLIBC_2.2) [2]	putc(GLIBC_2.2) [2]	snprintf(GLIBC_2.2) [2]	
fgetc(GLIBC_2.2) [2]	fseeko(GLIBC_2.2) [2]	putc_unlocked(GLIBC_2.2) [2]	sprintf(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

[3]. SUSv2

An LSB conforming implementation shall provide the architecture specific data interfaces for Standard I/O specified in Table 1-5, with the full functionality as described in the referenced underlying specification.

Table 1-5. libc - Standard I/O Data Interfaces

stderr(GLIBC_2.2) [1]	stdin(GLIBC_2.2) [1]	stdout(GLIBC_2.2) [1]		

Referenced Specification(s)

[1]. ISO POSIX (2003)

1.2.4. Signal Handling

1.2.4.1. Interfaces for Signal Handling

An LSB conforming implementation shall provide the architecture specific functions for Signal Handling specified in Table 1-6, with the full functionality as described in the referenced underlying specification.

Table 1-6. libc - Signal Handling Function Interfaces

__libc_current_sigrtmax(GLIBC_2.2) [1]	sigaddset(GLIBC_2.2) [2]	sighold(GLIBC_2.2) [2]	sigpause(GLIBC_2.2) [2]	sigsuspend(GLIBC_2.2) [2]
__libc_current_sigrt	sigaltstack(GLIBC_	sigignore(GLIBC_2	sigpending(GLIBC_	sigtimedwait(GLIB

min(GLIBC_2.2) [1]	2.2) [2]	.2) [2]	2.2) [2]	C_2.2) [2]
__sigsetjmp(GLIBC_2.2) [1]	sigandset(GLIBC_2.2) [1]	siginterrupt(GLIBC_2.2) [2]	sigprocmask(GLIBC_2.2) [2]	sigwait(GLIBC_2.2) [2]
__sysv_signal(GLIBC_2.2) [1]	sigblock(GLIBC_2.2) [1]	sigisemptyset(GLIBC_2.2) [1]	sigqueue(GLIBC_2.2) [2]	sigwaitinfo(GLIBC_2.2) [2]
bsd_signal(GLIBC_2.2) [2]	sigdelset(GLIBC_2.2) [2]	sigismember(GLIBC_2.2) [2]	sigrelse(GLIBC_2.2) [2]	
psignal(GLIBC_2.2) [1]	sigemptyset(GLIBC_2.2) [2]	siglongjmp(GLIBC_2.2) [2]	sigreturn(GLIBC_2.2) [1]	
raise(GLIBC_2.2) [2]	sigfillset(GLIBC_2.2) [2]	signal(GLIBC_2.2) [2]	sigset(GLIBC_2.2) [2]	
sigaction(GLIBC_2.2) [2]	siggetmask(GLIBC_2.2) [1]	sigorset(GLIBC_2.2) [1]	sigstack(GLIBC_2.2) [3]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

[3]. SUSv2

An LSB conforming implementation shall provide the architecture specific data interfaces for Signal Handling specified in Table 1-7, with the full functionality as described in the referenced underlying specification.

Table 1-7. libc - Signal Handling Data Interfaces

_sys_siglist(GLIBC_2.3.3) [1]				

Referenced Specification(s)

[1]. this specification

1.2.5. Localization Functions

1.2.5.1. Interfaces for Localization Functions

An LSB conforming implementation shall provide the architecture specific functions for Localization Functions specified in Table 1-8, with the full functionality as described in the referenced underlying specification.

Table 1-8. libc - Localization Functions Function Interfaces

bind_textdomain_codeset(GLIBC_2.2) [1]	catopen(GLIBC_2.2) [2]	dngettext(GLIBC_2.2) [1]	iconv_open(GLIBC_2.2) [2]	setlocale(GLIBC_2.2) [2]
bindtextdomain(GL	dcgettext(GLIBC_2.	gettext(GLIBC_2.2)	localeconv(GLIBC_	textdomain(GLIBC

IBC_2.2) [1]	2) [1]	[1]	2.2) [2]	_2.2) [1]
catclose(GLIBC_2.2) [2]	dcngettext(GLIBC_2.2) [1]	iconv(GLIBC_2.2) [2]	ngettext(GLIBC_2.2) [1]	
catgets(GLIBC_2.2) [2]	dgettext(GLIBC_2.2) [1]	iconv_close(GLIBC_2.2) [2]	nl_langinfo(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

An LSB conforming implementation shall provide the architecture specific data interfaces for Localization Functions specified in Table 1-9, with the full functionality as described in the referenced underlying specification.

Table 1-9. libc - Localization Functions Data Interfaces

_nl_msg_cat_cntr(GLIBC_2.2) [1]				

Referenced Specification(s)

[1]. this specification

1.2.6. Socket Interface

1.2.6.1. Interfaces for Socket Interface

An LSB conforming implementation shall provide the architecture specific functions for Socket Interface specified in Table 1-10, with the full functionality as described in the referenced underlying specification.

Table 1-10. libc - Socket Interface Function Interfaces

__h_errno_location(GLIBC_2.2) [1]	gethostid(GLIBC_2.2) [2]	listen(GLIBC_2.2) [2]	sendmsg(GLIBC_2.2) [2]	socketpair(GLIBC_2.2) [2]
accept(GLIBC_2.2) [2]	gethostname(GLIBC_2.2) [2]	recv(GLIBC_2.2) [2]	sendto(GLIBC_2.2) [2]	
bind(GLIBC_2.2) [2]	getpeername(GLIBC_2.2) [2]	recvfrom(GLIBC_2.2) [2]	setsockopt(GLIBC_2.2) [1]	
bindresvport(GLIBC_2.2) [1]	getsockname(GLIBC_2.2) [2]	recvmsg(GLIBC_2.2) [2]	shutdown(GLIBC_2.2) [2]	
connect(GLIBC_2.2) [2]	getsockopt(GLIBC_2.2) [2]	send(GLIBC_2.2) [2]	socket(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

An LSB conforming implementation shall provide the architecture specific deprecated functions for Socket Interface specified in Table 1-11, with the full functionality as described in the referenced underlying specification.

> These interfaces are deprecated, and applications should avoid using them. These interfaces may be withdrawn in future releases of this specification.

Table 1-11. libc - Socket Interface Deprecated Function Interfaces

gethostbyname_r(GLIBC_2.2) [1]				

Referenced Specification(s)

[1]. this specification

1.2.7. Wide Characters

1.2.7.1. Interfaces for Wide Characters

An LSB conforming implementation shall provide the architecture specific functions for Wide Characters specified in Table 1-12, with the full functionality as described in the referenced underlying specification.

Table 1-12. libc - Wide Characters Function Interfaces

__wcstod_internal(GLIBC_2.2) [1]	mbsinit(GLIBC_2.2) [2]	vwscanf(GLIBC_2.2) [2]	wcsnlen(GLIBC_2.2) [1]	wcstoumax(GLIBC_2.2) [2]
__wcstof_internal(GLIBC_2.2) [1]	mbsnrtowcs(GLIBC_2.2) [1]	wcpcpy(GLIBC_2.2) [1]	wcsnrtombs(GLIBC_2.2) [1]	wcstouq(GLIBC_2.2) [1]
__wcstol_internal(GLIBC_2.2) [1]	mbsrtowcs(GLIBC_2.2) [2]	wcpncpy(GLIBC_2.2) [1]	wcspbrk(GLIBC_2.2) [2]	wcswcs(GLIBC_2.2) [2]
__wcstold_internal(GLIBC_2.2) [1]	mbstowcs(GLIBC_2.2) [2]	wcrtomb(GLIBC_2.2) [2]	wcsrchr(GLIBC_2.2) [2]	wcswidth(GLIBC_2.2) [2]
__wcstoul_internal(GLIBC_2.2) [1]	mbtowc(GLIBC_2.2) [2]	wcscasecmp(GLIBC_2.2) [1]	wcsrtombs(GLIBC_2.2) [2]	wcsxfrm(GLIBC_2.2) [2]
btowc(GLIBC_2.2) [2]	putwc(GLIBC_2.2) [2]	wcscat(GLIBC_2.2) [2]	wcsspn(GLIBC_2.2) [2]	wctob(GLIBC_2.2) [2]
fgetwc(GLIBC_2.2) [2]	putwchar(GLIBC_2.2) [2]	wcschr(GLIBC_2.2) [2]	wcsstr(GLIBC_2.2) [2]	wctomb(GLIBC_2.2) [2]
fgetws(GLIBC_2.2) [2]	swprintf(GLIBC_2.2) [2]	wcscmp(GLIBC_2.2) [2]	wcstod(GLIBC_2.2) [2]	wctrans(GLIBC_2.2) [2]
fputwc(GLIBC_2.2) [2]	swscanf(GLIBC_2.2) [2]	wcscoll(GLIBC_2.2) [2]	wcstof(GLIBC_2.2) [2]	wctype(GLIBC_2.2) [2]
fputws(GLIBC_2.2) [2]	towctrans(GLIBC_2.2) [2]	wcscpy(GLIBC_2.2) [2]	wcstoimax(GLIBC_2.2) [2]	wcwidth(GLIBC_2.2) [2]

fwide(GLIBC_2.2) [2]	towlower(GLIBC_2.2) [2]	wcscspn(GLIBC_2.2) [2]	wcstok(GLIBC_2.2) [2]	wmemchr(GLIBC_2.2) [2]
fwprintf(GLIBC_2.2) [2]	towupper(GLIBC_2.2) [2]	wcsdup(GLIBC_2.2) [1]	wcstol(GLIBC_2.2) [2]	wmemcmp(GLIBC_2.2) [2]
fwscanf(GLIBC_2.2) [2]	ungetwc(GLIBC_2.2) [2]	wcsftime(GLIBC_2.2) [2]	wcstold(GLIBC_2.2) [2]	wmemcpy(GLIBC_2.2) [2]
getwc(GLIBC_2.2) [2]	vfwprintf(GLIBC_2.2) [2]	wcslen(GLIBC_2.2) [2]	wcstoll(GLIBC_2.2) [2]	wmemmove(GLIBC_2.2) [2]
getwchar(GLIBC_2.2) [2]	vfwscanf(GLIBC_2.2) [2]	wcsncasecmp(GLIBC_2.2) [1]	wcstombs(GLIBC_2.2) [2]	wmemset(GLIBC_2.2) [2]
mblen(GLIBC_2.2) [2]	vswprintf(GLIBC_2.2) [2]	wcsncat(GLIBC_2.2) [2]	wcstoq(GLIBC_2.2) [1]	wprintf(GLIBC_2.2) [2]
mbrlen(GLIBC_2.2) [2]	vswscanf(GLIBC_2.2) [2]	wcsncmp(GLIBC_2.2) [2]	wcstoul(GLIBC_2.2) [2]	wscanf(GLIBC_2.2) [2]
mbrtowc(GLIBC_2.2) [2]	vwprintf(GLIBC_2.2) [2]	wcsncpy(GLIBC_2.2) [2]	wcstoull(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

1.2.8. String Functions

1.2.8.1. Interfaces for String Functions

An LSB conforming implementation shall provide the architecture specific functions for String Functions specified in Table 1-13, with the full functionality as described in the referenced underlying specification.

Table 1-13. libc - String Functions Function Interfaces

__mempcpy(GLIBC_2.2) [1]	bzero(GLIBC_2.2) [2]	strcasestr(GLIBC_2.2) [1]	strncasecmp(GLIBC_2.2) [2]	strtoimax(GLIBC_2.2) [2]
__rawmemchr(GLIBC_2.2) [1]	ffs(GLIBC_2.2) [2]	strcat(GLIBC_2.2) [2]	strncat(GLIBC_2.2) [2]	strtok(GLIBC_2.2) [2]
__stpcpy(GLIBC_2.2) [1]	index(GLIBC_2.2) [2]	strchr(GLIBC_2.2) [2]	strncmp(GLIBC_2.2) [2]	strtok_r(GLIBC_2.2) [2]
__strdup(GLIBC_2.2) [1]	memccpy(GLIBC_2.2) [2]	strcmp(GLIBC_2.2) [2]	strncpy(GLIBC_2.2) [2]	strtold(GLIBC_2.2) [2]
__strtod_internal(GLIBC_2.2) [1]	memchr(GLIBC_2.2) [2]	strcoll(GLIBC_2.2) [2]	strndup(GLIBC_2.2) [1]	strtoll(GLIBC_2.2) [2]
__strtof_internal(G	memcmp(GLIBC_2	strcpy(GLIBC_2.2)	strnlen(GLIBC_2.2)	strtoq(GLIBC_2.2)

LIBC_2.2) [1]	.2) [2]	[2]	[1]	[1]
__strtok_r(GLIBC_2.2) [1]	memcpy(GLIBC_2.2) [2]	strcspn(GLIBC_2.2) [2]	strpbrk(GLIBC_2.2) [2]	strtoull(GLIBC_2.2) [2]
__strtol_internal(GLIBC_2.2) [1]	memmove(GLIBC_2.2) [2]	strdup(GLIBC_2.2) [2]	strptime(GLIBC_2.2) [1]	strtoumax(GLIBC_2.2) [2]
__strtold_internal(GLIBC_2.2) [1]	memrchr(GLIBC_2.2) [1]	strerror(GLIBC_2.2) [2]	strrchr(GLIBC_2.2) [2]	strtouq(GLIBC_2.2) [1]
__strtoll_internal(GLIBC_2.2) [1]	memset(GLIBC_2.2) [2]	strerror_r(GLIBC_2.2) [1]	strsep(GLIBC_2.2) [1]	strverscmp(GLIBC_2.2) [1]
__strtoul_internal(GLIBC_2.2) [1]	rindex(GLIBC_2.2) [2]	strfmon(GLIBC_2.2) [2]	strsignal(GLIBC_2.2) [1]	strxfrm(GLIBC_2.2) [2]
__strtoull_internal(GLIBC_2.2) [1]	stpcpy(GLIBC_2.2) [1]	strfry(GLIBC_2.2) [1]	strspn(GLIBC_2.2) [2]	swab(GLIBC_2.2) [2]
bcmp(GLIBC_2.2) [2]	stpncpy(GLIBC_2.2) [1]	strftime(GLIBC_2.2) [2]	strstr(GLIBC_2.2) [2]	
bcopy(GLIBC_2.2) [2]	strcasecmp(GLIBC_2.2) [2]	strlen(GLIBC_2.2) [2]	strtof(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

1.2.9. IPC Functions

1.2.9.1. Interfaces for IPC Functions

An LSB conforming implementation shall provide the architecture specific functions for IPC Functions specified in Table 1-14, with the full functionality as described in the referenced underlying specification.

Table 1-14. libc - IPC Functions Function Interfaces

ftok(GLIBC_2.2) [1]	msgrcv(GLIBC_2.2) [1]	semget(GLIBC_2.2) [1]	shmctl(GLIBC_2.2) [1]	
msgctl(GLIBC_2.2) [1]	msgsnd(GLIBC_2.2) [1]	semop(GLIBC_2.2) [1]	shmdt(GLIBC_2.2) [1]	
msgget(GLIBC_2.2) [1]	semctl(GLIBC_2.2) [1]	shmat(GLIBC_2.2) [1]	shmget(GLIBC_2.2) [1]	

Referenced Specification(s)

[1]. ISO POSIX (2003)

1.2.10. Regular Expressions

1.2.10.1. Interfaces for Regular Expressions

An LSB conforming implementation shall provide the architecture specific functions for Regular Expressions specified in Table 1-15, with the full functionality as described in the referenced underlying specification.

Table 1-15. libc - Regular Expressions Function Interfaces

regcomp(GLIBC_2.2) [1]	regerror(GLIBC_2.2) [1]	regexec(GLIBC_2.2) [1]	regfree(GLIBC_2.2) [1]	

Referenced Specification(s)

[1]. ISO POSIX (2003)

An LSB conforming implementation shall provide the architecture specific deprecated functions for Regular Expressions specified in Table 1-16, with the full functionality as described in the referenced underlying specification.

These interfaces are deprecated, and applications should avoid using them. These interfaces may be withdrawn in future releases of this specification.

Table 1-16. libc - Regular Expressions Deprecated Function Interfaces

advance(GLIBC_2.2) [1]	re_comp(GLIBC_2.2) [1]	re_exec(GLIBC_2.2) [1]	step(GLIBC_2.2) [1]	

Referenced Specification(s)

[1]. SUSv2

An LSB conforming implementation shall provide the architecture specific deprecated data interfaces for Regular Expressions specified in Table 1-17, with the full functionality as described in the referenced underlying specification.

These interfaces are deprecated, and applications should avoid using them. These interfaces may be withdrawn in future releases of this specification.

Table 1-17. libc - Regular Expressions Deprecated Data Interfaces

loc1(GLIBC_2.2) [1]	loc2(GLIBC_2.2) [1]	locs(GLIBC_2.2) [1]		

Referenced Specification(s)

[1]. SUSv2

1.2.11. Character Type Functions

1.2.11.1. Interfaces for Character Type Functions

An LSB conforming implementation shall provide the architecture specific functions for Character Type Functions specified in Table 1-18, with the full functionality as described in the referenced underlying specification.

Table 1-18. libc - Character Type Functions Function Interfaces

__ctype_get_mb_cur_max(GLIBC_2.2) [1]	isdigit(GLIBC_2.2) [2]	iswalnum(GLIBC_2.2) [2]	iswlower(GLIBC_2.2) [2]	toascii(GLIBC_2.2) [2]
_tolower(GLIBC_2.2) [2]	isgraph(GLIBC_2.2) [2]	iswalpha(GLIBC_2.2) [2]	iswprint(GLIBC_2.2) [2]	tolower(GLIBC_2.2) [2]
_toupper(GLIBC_2.2) [2]	islower(GLIBC_2.2) [2]	iswblank(GLIBC_2.2) [2]	iswpunct(GLIBC_2.2) [2]	toupper(GLIBC_2.2) [2]
isalnum(GLIBC_2.2) [2]	isprint(GLIBC_2.2) [2]	iswcntrl(GLIBC_2.2) [2]	iswspace(GLIBC_2.2) [2]	
isalpha(GLIBC_2.2) [2]	ispunct(GLIBC_2.2) [2]	iswctype(GLIBC_2.2) [2]	iswupper(GLIBC_2.2) [2]	
isascii(GLIBC_2.2) [2]	isspace(GLIBC_2.2) [2]	iswdigit(GLIBC_2.2) [2]	iswxdigit(GLIBC_2.2) [2]	
iscntrl(GLIBC_2.2) [2]	isupper(GLIBC_2.2) [2]	iswgraph(GLIBC_2.2) [2]	isxdigit(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

1.2.12. Time Manipulation

1.2.12.1. Interfaces for Time Manipulation

An LSB conforming implementation shall provide the architecture specific functions for Time Manipulation specified in Table 1-19, with the full functionality as described in the referenced underlying specification.

Table 1-19. libc - Time Manipulation Function Interfaces

adjtime(GLIBC_2.2) [1]	ctime(GLIBC_2.2) [2]	gmtime(GLIBC_2.2) [2]	localtime_r(GLIBC_2.2) [2]	ualarm(GLIBC_2.2) [2]
asctime(GLIBC_2.2) [2]	ctime_r(GLIBC_2.2) [2]	gmtime_r(GLIBC_2.2) [2]	mktime(GLIBC_2.2) [2]	
asctime_r(GLIBC_2.2) [2]	difftime(GLIBC_2.2) [2]	localtime(GLIBC_2.2) [2]	tzset(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

An LSB conforming implementation shall provide the architecture specific deprecated functions for Time Manipulation specified in Table 1-20, with the full functionality as described in the referenced underlying specification.

These interfaces are deprecated, and applications should avoid using them. These interfaces may be withdrawn in future releases of this specification.

Table 1-20. libc - Time Manipulation Deprecated Function Interfaces

adjtimex(GLIBC_2.2) [1]				

Referenced Specification(s)

[1]. this specification

An LSB conforming implementation shall provide the architecture specific data interfaces for Time Manipulation specified in Table 1-21, with the full functionality as described in the referenced underlying specification.

Table 1-21. libc - Time Manipulation Data Interfaces

__daylight(GLIBC_2.2) [1]	__tzname(GLIBC_2.2) [1]	timezone(GLIBC_2.2) [2]		
__timezone(GLIBC_2.2) [1]	daylight(GLIBC_2.2) [2]	tzname(GLIBC_2.2) [2]		

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

1.2.13. Terminal Interface Functions

1.2.13.1. Interfaces for Terminal Interface Functions

An LSB conforming implementation shall provide the architecture specific functions for Terminal Interface Functions specified in Table 1-22, with the full functionality as described in the referenced underlying specification.

Table 1-22. libc - Terminal Interface Functions Function Interfaces

cfgetispeed(GLIBC_2.2) [1]	cfsetispeed(GLIBC_2.2) [1]	tcdrain(GLIBC_2.2) [1]	tcgetattr(GLIBC_2.2) [1]	tcsendbreak(GLIBC_2.2) [1]
cfgetospeed(GLIBC_2.2) [1]	cfsetospeed(GLIBC_2.2) [1]	tcflow(GLIBC_2.2) [1]	tcgetpgrp(GLIBC_2.2) [1]	tcsetattr(GLIBC_2.2) [1]
cfmakeraw(GLIBC_2.2) [2]	cfsetspeed(GLIBC_2.2) [2]	tcflush(GLIBC_2.2) [1]	tcgetsid(GLIBC_2.2) [1]	tcsetpgrp(GLIBC_2.2) [1]

Referenced Specification(s)

[1]. ISO POSIX (2003)

[2]. this specification

1.2.14. System Database Interface

1.2.14.1. Interfaces for System Database Interface

An LSB conforming implementation shall provide the architecture specific functions for System Database Interface specified in Table 1-23, with the full functionality as described in the referenced underlying specification.

Table 1-23. libc - System Database Interface Function Interfaces

endgrent(GLIBC_2.2) [1]	getgrgid(GLIBC_2.2) [1]	getprotobynumber(GLIBC_2.2) [1]	getservbyport(GLIBC_2.2) [1]	setgrent(GLIBC_2.2) [1]
endnetent(GLIBC_2.2) [1]	getgrgid_r(GLIBC_2.2) [1]	getprotoent(GLIBC_2.2) [1]	getservent(GLIBC_2.2) [1]	setgroups(GLIBC_2.2) [2]
endprotoent(GLIBC_2.2) [1]	getgrnam(GLIBC_2.2) [1]	getpwent(GLIBC_2.2) [1]	getutent(GLIBC_2.2) [2]	setnetent(GLIBC_2.2) [1]
endpwent(GLIBC_2.2) [1]	getgrnam_r(GLIBC_2.2) [1]	getpwnam(GLIBC_2.2) [1]	getutent_r(GLIBC_2.2) [2]	setprotoent(GLIBC_2.2) [1]
endservent(GLIBC_2.2) [1]	gethostbyaddr(GLIBC_2.2) [1]	getpwnam_r(GLIBC_2.2) [1]	getutxent(GLIBC_2.2) [1]	setpwent(GLIBC_2.2) [1]
endutent(GLIBC_2.2) [3]	gethostbyname(GLIBC_2.2) [1]	getpwuid(GLIBC_2.2) [1]	getutxid(GLIBC_2.2) [1]	setservent(GLIBC_2.2) [1]
endutxent(GLIBC_2.2) [1]	getnetbyaddr(GLIBC_2.2) [1]	getpwuid_r(GLIBC_2.2) [1]	getutxline(GLIBC_2.2) [1]	setutent(GLIBC_2.2) [2]
getgrent(GLIBC_2.2) [1]	getprotobyname(GLIBC_2.2) [1]	getservbyname(GLIBC_2.2) [1]	pututxline(GLIBC_2.2) [1]	setutxent(GLIBC_2.2) [1]

Referenced Specification(s)

[1]. ISO POSIX (2003)

[2]. this specification

[3]. SUSv2

1.2.15. Language Support

1.2.15.1. Interfaces for Language Support

An LSB conforming implementation shall provide the architecture specific functions for Language Support specified in Table 1-24, with the full functionality as described in the referenced underlying specification.

Table 1-24. libc - Language Support Function Interfaces

__libc_start_main(GLIBC_2.2) [1]	_obstack_begin(GLIBC_2.2) [1]	_obstack_newchunk(GLIBC_2.2) [1]	obstack_free(GLIBC_2.2) [1]	

Referenced Specification(s)

[1]. this specification

1.2.16. Large File Support

1.2.16.1. Interfaces for Large File Support

An LSB conforming implementation shall provide the architecture specific functions for Large File Support specified in Table 1-25, with the full functionality as described in the referenced underlying specification.

Table 1-25. libc - Large File Support Function Interfaces

__fxstat64(GLIBC_2.2) [1]	fopen64(GLIBC_2.2) [2]	ftello64(GLIBC_2.2) [2]	lseek64(GLIBC_2.2) [2]	readdir64(GLIBC_2.2) [2]
__lxstat64(GLIBC_2.2) [1]	freopen64(GLIBC_2.2) [2]	ftruncate64(GLIBC_2.2) [2]	mkstemp64(GLIBC_2.2) [2]	statvfs64(GLIBC_2.2) [2]
__xstat64(GLIBC_2.2) [1]	fseeko64(GLIBC_2.2) [2]	ftw64(GLIBC_2.2) [2]	mmap64(GLIBC_2.2) [2]	tmpfile64(GLIBC_2.2) [2]
creat64(GLIBC_2.2) [2]	fsetpos64(GLIBC_2.2) [2]	getrlimit64(GLIBC_2.2) [2]	nftw64(GLIBC_2.2) [2]	truncate64(GLIBC_2.2) [2]
fgetpos64(GLIBC_2.2) [2]	fstatvfs64(GLIBC_2.2) [2]	lockf64(GLIBC_2.2) [2]	open64(GLIBC_2.2) [2]	

Referenced Specification(s)

[1]. this specification

[2]. Large File Support

1.2.17. Standard Library

1.2.17.1. Interfaces for Standard Library

An LSB conforming implementation shall provide the architecture specific functions for Standard Library specified in Table 1-26, with the full functionality as described in the referenced underlying specification.

Table 1-26. libc - Standard Library Function Interfaces

_Exit(GLIBC_2.2) [1]	dirname(GLIBC_2.2) [1]	glob(GLIBC_2.2) [1]	lsearch(GLIBC_2.2) [1]	srand(GLIBC_2.2) [1]
__assert_fail(GLIBC_2.2) [2]	div(GLIBC_2.2) [1]	glob64(GLIBC_2.2) [2]	makecontext(GLIBC_2.2) [1]	srand48(GLIBC_2.2) [1]
__cxa_atexit(GLIBC_2.2) [2]	drand48(GLIBC_2.2) [1]	globfree(GLIBC_2.2) [1]	malloc(GLIBC_2.2) [1]	srandom(GLIBC_2.2) [1]
__errno_location(GLIBC_2.2) [2]	ecvt(GLIBC_2.2) [1]	globfree64(GLIBC_2.2) [2]	memmem(GLIBC_2.2) [2]	strtod(GLIBC_2.2) [1]

__fpending(GLIBC_2.2) [2]	erand48(GLIBC_2.2) [1]	grantpt(GLIBC_2.2) [1]	mkstemp(GLIBC_2.2) [1]	strtol(GLIBC_2.2) [1]
__getpagesize(GLIBC_2.2) [2]	err(GLIBC_2.2) [2]	hcreate(GLIBC_2.2) [1]	mktemp(GLIBC_2.2) [1]	strtoul(GLIBC_2.2) [1]
__isinf(GLIBC_2.2) [2]	error(GLIBC_2.2) [2]	hdestroy(GLIBC_2.2) [1]	mrand48(GLIBC_2.2) [1]	swapcontext(GLIBC_2.2) [1]
__isinff(GLIBC_2.2) [2]	errx(GLIBC_2.2) [2]	hsearch(GLIBC_2.2) [1]	nftw(GLIBC_2.2) [1]	syslog(GLIBC_2.2) [1]
__isinfl(GLIBC_2.2) [2]	fcvt(GLIBC_2.2) [1]	htonl(GLIBC_2.2) [1]	nrand48(GLIBC_2.2) [1]	system(GLIBC_2.2) [2]
__isnan(GLIBC_2.2) [2]	fmtmsg(GLIBC_2.2) [1]	htons(GLIBC_2.2) [1]	ntohl(GLIBC_2.2) [1]	tdelete(GLIBC_2.2) [1]
__isnanf(GLIBC_2.2) [2]	fnmatch(GLIBC_2.2.3) [1]	imaxabs(GLIBC_2.2) [1]	ntohs(GLIBC_2.2) [1]	tfind(GLIBC_2.2) [1]
__isnanl(GLIBC_2.2) [2]	fpathconf(GLIBC_2.2) [1]	imaxdiv(GLIBC_2.2) [1]	openlog(GLIBC_2.2) [1]	tmpfile(GLIBC_2.2) [1]
__sysconf(GLIBC_2.2) [2]	free(GLIBC_2.2) [1]	inet_addr(GLIBC_2.2) [1]	perror(GLIBC_2.2) [1]	tmpnam(GLIBC_2.2) [1]
_exit(GLIBC_2.2) [1]	freeaddrinfo(GLIBC_2.2) [1]	inet_ntoa(GLIBC_2.2) [1]	posix_memalign(GLIBC_2.2) [1]	tsearch(GLIBC_2.2) [1]
_longjmp(GLIBC_2.2) [1]	ftrylockfile(GLIBC_2.2) [1]	inet_ntop(GLIBC_2.2) [1]	ptsname(GLIBC_2.2) [1]	ttyname(GLIBC_2.2) [1]
_setjmp(GLIBC_2.2) [1]	ftw(GLIBC_2.2) [1]	inet_pton(GLIBC_2.2) [1]	putenv(GLIBC_2.2) [1]	ttyname_r(GLIBC_2.2) [1]
a64l(GLIBC_2.2) [1]	funlockfile(GLIBC_2.2) [1]	initstate(GLIBC_2.2) [1]	qsort(GLIBC_2.2) [1]	twalk(GLIBC_2.2) [1]
abort(GLIBC_2.2) [1]	gai_strerror(GLIBC_2.2) [1]	insque(GLIBC_2.2) [1]	rand(GLIBC_2.2) [1]	unlockpt(GLIBC_2.2) [1]
abs(GLIBC_2.2) [1]	gcvt(GLIBC_2.2) [1]	isatty(GLIBC_2.2) [1]	rand_r(GLIBC_2.2) [1]	unsetenv(GLIBC_2.2) [1]
atof(GLIBC_2.2) [1]	getaddrinfo(GLIBC_2.2) [1]	isblank(GLIBC_2.2) [1]	random(GLIBC_2.2) [1]	usleep(GLIBC_2.2) [1]
atoi(GLIBC_2.2) [1]	getcwd(GLIBC_2.2) [1]	jrand48(GLIBC_2.2) [1]	random_r(GLIBC_2.2) [2]	verrx(GLIBC_2.2) [2]
atol(GLIBC_2.2) [1]	getdate(GLIBC_2.2) [1]	l64a(GLIBC_2.2) [1]	realloc(GLIBC_2.2) [1]	vfscanf(GLIBC_2.2) [1]
atoll(GLIBC_2.2)	getenv(GLIBC_2.2)	labs(GLIBC_2.2)	realpath(GLIBC_2.	vscanf(GLIBC_2.2)

[1]	[1]	[1]	3) [1]	[1]
basename(GLIBC_2.2) [1]	getlogin(GLIBC_2.2) [1]	lcong48(GLIBC_2.2) [1]	remque(GLIBC_2.2) [1]	vsscanf(GLIBC_2.2) [1]
bsearch(GLIBC_2.2) [1]	getnameinfo(GLIBC_2.2) [1]	ldiv(GLIBC_2.2) [1]	seed48(GLIBC_2.2) [1]	vsyslog(GLIBC_2.2) [2]
calloc(GLIBC_2.2) [1]	getopt(GLIBC_2.2) [2]	lfind(GLIBC_2.2) [1]	setenv(GLIBC_2.2) [1]	warn(GLIBC_2.2) [2]
closelog(GLIBC_2.2) [1]	getopt_long(GLIBC_2.2) [2]	llabs(GLIBC_2.2) [1]	sethostid(GLIBC_2.2) [2]	warnx(GLIBC_2.2) [2]
confstr(GLIBC_2.2) [1]	getopt_long_only(GLIBC_2.2) [2]	lldiv(GLIBC_2.2) [1]	sethostname(GLIBC_2.2) [2]	wordexp(GLIBC_2.2) [1]
cuserid(GLIBC_2.2) [3]	getsubopt(GLIBC_2.2) [1]	longjmp(GLIBC_2.2) [1]	setlogmask(GLIBC_2.2) [1]	wordfree(GLIBC_2.2) [1]
daemon(GLIBC_2.2) [2]	gettimeofday(GLIBC_2.2) [1]	lrand48(GLIBC_2.2) [1]	setstate(GLIBC_2.2) [1]	

Referenced Specification(s)

[1]. ISO POSIX (2003)

[2]. this specification

[3]. SUSv2

An LSB conforming implementation shall provide the architecture specific data interfaces for Standard Library specified in Table 1-27, with the full functionality as described in the referenced underlying specification.

Table 1-27. libc - Standard Library Data Interfaces

__environ(GLIBC_2.2) [1]	_sys_errlist(GLIBC_2.3) [1]	getdate_err(GLIBC_2.2) [2]	opterr(GLIBC_2.2) [1]	optopt(GLIBC_2.2) [1]
_environ(GLIBC_2.2) [1]	environ(GLIBC_2.2) [2]	optarg(GLIBC_2.2) [2]	optind(GLIBC_2.2) [1]	

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

1.3. Data Definitions for libc

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.

These definitions are intended to supplement those provided in the referenced underlying specifications.

This specification uses ISO/IEC 9899 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.

1.3.1. errno.h

```
#define EDEADLOCK       EDEADLK
```

1.3.2. inttypes.h

```
typedef long intmax_t;
typedef unsigned long uintmax_t;
typedef unsigned long uintptr_t;
typedef unsigned long uint64_t;
```

1.3.3. limits.h

```
#define LONG_MAX        0x7FFFFFFFFFFFFFFFL
#define ULONG_MAX       0xFFFFFFFFFFFFFFFFUL

#define CHAR_MAX        SCHAR_MAX
#define CHAR_MIN        SCHAR_MIN
```

1.3.4. setjmp.h

```
typedef long __jmp_buf[70] __attribute__ ((aligned (16)));
```

1.3.5. signal.h

```
struct sigaction
{
  union
  {
    sighandler_t _sa_handler;
    void (*_sa_sigaction) (int, siginfo_t *, void *);
  }
  __sigaction_handler;
  unsigned long sa_flags;
  sigset_t sa_mask;
}
 ;
#define MINSIGSTKSZ     131027
#define SIGSTKSZ        262144

struct ia64_fpreg
{
```

```
  union
  {
    unsigned long bits[2];
    long double __dummy;
  }
  u;
}
 ;

struct sigcontext
{
  unsigned long sc_flags;
  unsigned long sc_nat;
  stack_t sc_stack;
  unsigned long sc_ip;
  unsigned long sc_cfm;
  unsigned long sc_um;
  unsigned long sc_ar_rsc;
  unsigned long sc_ar_bsp;
  unsigned long sc_ar_rnat;
  unsigned long sc_ar_ccv;
  unsigned long sc_ar_unat;
  unsigned long sc_ar_fpsr;
  unsigned long sc_ar_pfs;
  unsigned long sc_ar_lc;
  unsigned long sc_pr;
  unsigned long sc_br[8];
  unsigned long sc_gr[32];
  struct ia64_fpreg sc_fr[128];
  unsigned long sc_rbs_base;
  unsigned long sc_loadrs;
  unsigned long sc_ar25;
  unsigned long sc_ar26;
  unsigned long sc_rsvd[12];
  unsigned long sc_mask;
}
 ;
```

1.3.6. stddef.h

```

typedef long ptrdiff_t;
typedef unsigned long size_t;
```

1.3.7. sys/ioctl.h

```

#define FIONREAD        0x541B
#define TIOCNOTTY       0x5422
```

1.3.8. sys/ipc.h

```
struct ipc_perm
{
  key_t __key;
  uid_t uid;
  gid_t gid;
  uid_t cuid;
  uid_t cgid;
  mode_t mode;
  unsigned short __seq;
  unsigned short __pad1;
  unsigned long __unused1;
  unsigned long __unused2;
}
 ;
```

1.3.9. sys/mman.h

```
#define MCL_CURRENT     1
#define MCL_FUTURE      2
```

1.3.10. sys/msg.h

```
struct msqid_ds
{
  struct ipc_perm msg_perm;
  time_t msg_stime;
  time_t msg_rtime;
  time_t msg_ctime;
  unsigned long __msg_cbytes;
  unsigned long msg_qnum;
  unsigned long msg_qbytes;
  pid_t msg_lspid;
  pid_t msg_lrpid;
  unsigned long __unused1;
  unsigned long __unused2;
}
 ;
```

1.3.11. sys/sem.h

```
struct semid_ds
{
  struct ipc_perm sem_perm;
  time_t sem_otime;
  time_t sem_ctime;
```

```
  unsigned long sem_nsems;
  unsigned long __unused1;
  unsigned long __unused2;
}
 ;
```

1.3.12. sys/shm.h

```

#define SHMLBA  (1024*1024)

struct shmid_ds
{
  struct ipc_perm shm_perm;
  size_t shm_segsz;
  time_t shm_atime;
  time_t shm_dtime;
  time_t shm_ctime;
  pid_t shm_cpid;
  pid_t shm_lpid;
  unsigned long shm_nattch;
  unsigned long __unused1;
  unsigned long __unused2;
}
 ;
```

1.3.13. sys/socket.h

```

typedef uint64_t __ss_aligntype;
```

1.3.14. sys/stat.h

```

#define _STAT_VER       1

struct stat
{
  dev_t st_dev;
  ino_t st_ino;
  nlink_t st_nlink;
  mode_t st_mode;
  uid_t st_uid;
  gid_t st_gid;
  unsigned int pad0;
  dev_t st_rdev;
  off_t st_size;
  struct timespec st_atim;
  struct timespec st_mtim;
  struct timespec st_ctim;
  blksize_t st_blksize;
  blkcnt_t st_blocks;
```

```
  unsigned long __unused[3];
}
 ;
struct stat64
{
  dev_t st_dev;
  ino64_t st_ino;
  nlink_t st_nlink;
  mode_t st_mode;
  uid_t st_uid;
  gid_t st_gid;
  unsigned int pad0;
  dev_t st_rdev;
  off_t st_size;
  struct timespec st_atim;
  struct timespec st_mtim;
  struct timespec st_ctim;
  blksize_t st_blksize;
  blkcnt64_t st_blocks;
  unsigned long __unused[3];
}
 ;
```

1.3.15. sys/statvfs.h

```

struct statvfs
{
  unsigned long f_bsize;
  unsigned long f_frsize;
  fsblkcnt64_t f_blocks;
  fsblkcnt64_t f_bfree;
  fsblkcnt64_t f_bavail;
  fsfilcnt64_t f_files;
  fsfilcnt64_t f_ffree;
  fsfilcnt64_t f_favail;
  unsigned long f_fsid;
  unsigned long f_flag;
  unsigned long f_namemax;
  unsigned int __f_spare[6];
}
 ;
struct statvfs64
{
  unsigned long f_bsize;
  unsigned long f_frsize;
  fsblkcnt64_t f_blocks;
  fsblkcnt64_t f_bfree;
  fsblkcnt64_t f_bavail;
  fsfilcnt64_t f_files;
  fsfilcnt64_t f_ffree;
  fsfilcnt64_t f_favail;
```

```
  unsigned long f_fsid;
  unsigned long f_flag;
  unsigned long f_namemax;
  unsigned int __f_spare[6];
}
 ;
```

1.3.16. sys/types.h

```
typedef long int64_t;

typedef int64_t ssize_t;
```

1.3.17. termios.h

```
#define OLCUC   0000002
#define ONLCR   0000004
#define XCASE   0000004
#define NLDLY   0000400
#define CR1     0001000
#define IUCLC   0001000
#define CR2     0002000
#define CR3     0003000
#define CRDLY   0003000
#define TAB1    0004000
#define TAB2    0010000
#define TAB3    0014000
#define TABDLY  0014000
#define BS1     0020000
#define BSDLY   0020000
#define VT1     0040000
#define VTDLY   0040000
#define FF1     0100000
#define FFDLY   0100000

#define VSUSP   10
#define VEOL    11
#define VREPRINT        12
#define VDISCARD        13
#define VWERASE 14
#define VEOL2   16
#define VMIN    6
#define VSWTC   7
#define VSTART  8
#define VSTOP   9

#define IXON    0002000
#define IXOFF   0010000

#define CS6     0000020
```

```
#define CS7     0000040
#define CS8     0000060
#define CSIZE   0000060
#define CSTOPB  0000100
#define CREAD   0000200
#define PARENB  0000400
#define PARODD  0001000
#define HUPCL   0002000
#define CLOCAL  0004000
#define VTIME   5

#define ISIG    0000001
#define ICANON  0000002
#define ECHOE   0000020
#define ECHOK   0000040
#define ECHONL  0000100
#define NOFLSH  0000200
#define TOSTOP  0000400
#define ECHOCTL 0001000
#define ECHOPRT 0002000
#define ECHOKE  0004000
#define FLUSHO  0010000
#define PENDIN  0040000
#define IEXTEN  0100000
```

1.3.18. ucontext.h

```

#define _SC_GR0_OFFSET  (((char *) & ((struct sigcontext *) 0)->sc_gr[0]) - (char *) 0)

typedef struct sigcontext mcontext_t;

typedef struct ucontext
{
  union
  {
    mcontext_t _mc;
    struct
    {
      unsigned long _pad[_SC_GR0_OFFSET / 8];
      struct ucontext *_link;
    }
    _uc;
  }
  _u;
}
ucontext_t;
```

1.3.19. unistd.h

```

typedef long intptr_t;
```

1.3.20. utmp.h

```

struct lastlog
{
  time_t ll_time;
  char ll_line[UT_LINESIZE];
  char ll_host[UT_HOSTSIZE];
}
 ;

struct utmp
{
  short ut_type;
  pid_t ut_pid;
  char ut_line[UT_LINESIZE];
  char ut_id[4];
  char ut_user[UT_NAMESIZE];
  char ut_host[UT_HOSTSIZE];
  struct exit_status ut_exit;
  long ut_session;
  struct timeval ut_tv;
  int32_t ut_addr_v6[4];
  char __unused[20];
}
 ;
```

1.3.21. utmpx.h

```

struct utmpx
{
  short ut_type;
  pid_t ut_pid;
  char ut_line[UT_LINESIZE];
  char ut_id[4];
  char ut_user[UT_NAMESIZE];
  char ut_host[UT_HOSTSIZE];
  struct exit_status ut_exit;
  long ut_session;
  struct timeval ut_tv;
  int32_t ut_addr_v6[4];
  char __unused[20];
}
 ;
```

1.4. Interfaces for libm

Table 1-28 defines the library name and shared object name for the libm library

Table 1-28. libm Definition

Library:	libm
SONAME:	libm.so.6.1

The behavior of the interfaces in this library is specified by the following specifications:

ISO C (1999)
SUSv2
ISO POSIX (2003)

1.4.1. Math

1.4.1.1. Interfaces for Math

An LSB conforming implementation shall provide the architecture specific functions for Math specified in Table 1-29, with the full functionality as described in the referenced underlying specification.

Table 1-29. libm - Math Function Interfaces

acos(GLIBC_2.2) [1]	cexp(GLIBC_2.2) [1]	expf(GLIBC_2.2) [1]	jnf(GLIBC_2.2) [2]	remquof(GLIBC_2.2) [1]
acosf(GLIBC_2.2) [1]	cexpf(GLIBC_2.2) [1]	expl(GLIBC_2.2) [1]	jnl(GLIBC_2.2) [2]	remquol(GLIBC_2.2) [1]
acosh(GLIBC_2.2) [1]	cexpl(GLIBC_2.2) [1]	expm1(GLIBC_2.2) [1]	ldexp(GLIBC_2.2) [1]	rint(GLIBC_2.2) [1]
acoshf(GLIBC_2.2) [1]	cimag(GLIBC_2.2) [1]	fabs(GLIBC_2.2) [1]	ldexpf(GLIBC_2.2) [1]	rintf(GLIBC_2.2) [1]
acoshl(GLIBC_2.2) [1]	cimagf(GLIBC_2.2) [1]	fabsf(GLIBC_2.2) [1]	ldexpl(GLIBC_2.2) [1]	rintl(GLIBC_2.2) [1]
acosl(GLIBC_2.2) [1]	cimagl(GLIBC_2.2) [1]	fabsl(GLIBC_2.2) [1]	lgamma(GLIBC_2.2) [1]	round(GLIBC_2.2) [1]
asin(GLIBC_2.2) [1]	clog(GLIBC_2.2) [1]	fdim(GLIBC_2.2) [1]	lgamma_r(GLIBC_2.2) [2]	roundf(GLIBC_2.2) [1]
asinf(GLIBC_2.2) [1]	clog10(GLIBC_2.2) [2]	fdimf(GLIBC_2.2) [1]	lgammaf(GLIBC_2.2) [1]	roundl(GLIBC_2.2) [1]
asinh(GLIBC_2.2) [1]	clog10f(GLIBC_2.2) [2]	fdiml(GLIBC_2.2) [1]	lgammaf_r(GLIBC_2.2) [2]	scalb(GLIBC_2.2) [1]
asinhf(GLIBC_2.2) [1]	clog10l(GLIBC_2.2) [2]	feclearexcept(GLIBC_2.2) [1]	lgammal(GLIBC_2.2) [1]	scalbf(GLIBC_2.2) [2]
asinhl(GLIBC_2.2) [1]	clogf(GLIBC_2.2) [1]	fegetenv(GLIBC_2.2) [1]	lgammal_r(GLIBC_2.2) [2]	scalbl(GLIBC_2.2) [2]
asinl(GLIBC_2.2)	clogl(GLIBC_2.2)	fegetexceptflag(GLI	llrint(GLIBC_2.2)	scalbln(GLIBC_2.2)

[1]	[1]	BC_2.2) [1]	[1]	[1]
atan(GLIBC_2.2) [1]	conj(GLIBC_2.2) [1]	fegetround(GLIBC_2.2) [1]	llrintf(GLIBC_2.2) [1]	scalblnf(GLIBC_2.2) [1]
atan2(GLIBC_2.2) [1]	conjf(GLIBC_2.2) [1]	feholdexcept(GLIBC_2.2) [1]	llrintl(GLIBC_2.2) [1]	scalblnl(GLIBC_2.2) [1]
atan2f(GLIBC_2.2) [1]	conjl(GLIBC_2.2) [1]	feraiseexcept(GLIBC_2.2) [1]	llround(GLIBC_2.2) [1]	scalbn(GLIBC_2.2) [1]
atan2l(GLIBC_2.2) [1]	copysign(GLIBC_2.2) [1]	fesetenv(GLIBC_2.2) [1]	llroundf(GLIBC_2.2) [1]	scalbnf(GLIBC_2.2) [1]
atanf(GLIBC_2.2) [1]	copysignf(GLIBC_2.2) [1]	fesetexceptflag(GLIBC_2.2) [1]	llroundl(GLIBC_2.2) [1]	scalbnl(GLIBC_2.2) [1]
atanh(GLIBC_2.2) [1]	copysignl(GLIBC_2.2) [1]	fesetround(GLIBC_2.2) [1]	log(GLIBC_2.2) [1]	significand(GLIBC_2.2) [2]
atanhf(GLIBC_2.2) [1]	cos(GLIBC_2.2) [1]	fetestexcept(GLIBC_2.2) [1]	log10(GLIBC_2.2) [1]	significandf(GLIBC_2.2) [2]
atanhl(GLIBC_2.2) [1]	cosf(GLIBC_2.2) [1]	feupdateenv(GLIBC_2.2) [1]	log10f(GLIBC_2.2) [1]	significandl(GLIBC_2.2) [2]
atanl(GLIBC_2.2) [1]	cosh(GLIBC_2.2) [1]	finite(GLIBC_2.2) [3]	log10l(GLIBC_2.2) [1]	sin(GLIBC_2.2) [1]
cabs(GLIBC_2.2) [1]	coshf(GLIBC_2.2) [1]	finitef(GLIBC_2.2) [2]	log1p(GLIBC_2.2) [1]	sincos(GLIBC_2.2) [2]
cabsf(GLIBC_2.2) [1]	coshl(GLIBC_2.2) [1]	finitel(GLIBC_2.2) [2]	logb(GLIBC_2.2) [1]	sincosf(GLIBC_2.2) [2]
cabsl(GLIBC_2.2) [1]	cosl(GLIBC_2.2) [1]	floor(GLIBC_2.2) [1]	logf(GLIBC_2.2) [1]	sincosl(GLIBC_2.2) [2]
cacos(GLIBC_2.2) [1]	cpow(GLIBC_2.2) [1]	floorf(GLIBC_2.2) [1]	logl(GLIBC_2.2) [1]	sinf(GLIBC_2.2) [1]
cacosf(GLIBC_2.2) [1]	cpowf(GLIBC_2.2) [1]	floorl(GLIBC_2.2) [1]	lrint(GLIBC_2.2) [1]	sinh(GLIBC_2.2) [1]
cacosh(GLIBC_2.2) [1]	cpowl(GLIBC_2.2) [1]	fma(GLIBC_2.2) [1]	lrintf(GLIBC_2.2) [1]	sinhf(GLIBC_2.2) [1]
cacoshf(GLIBC_2.2) [1]	cproj(GLIBC_2.2) [1]	fmaf(GLIBC_2.2) [1]	lrintl(GLIBC_2.2) [1]	sinhl(GLIBC_2.2) [1]
cacoshl(GLIBC_2.2) [1]	cprojf(GLIBC_2.2) [1]	fmal(GLIBC_2.2) [1]	lround(GLIBC_2.2) [1]	sinl(GLIBC_2.2) [1]
cacosl(GLIBC_2.2) [1]	cprojl(GLIBC_2.2) [1]	fmax(GLIBC_2.2) [1]	lroundf(GLIBC_2.2) [1]	sqrt(GLIBC_2.2) [1]

carg(GLIBC_2.2) [1]	creal(GLIBC_2.2) [1]	fmaxf(GLIBC_2.2) [1]	lroundl(GLIBC_2.2) [1]	sqrtf(GLIBC_2.2) [1]
cargf(GLIBC_2.2) [1]	crealf(GLIBC_2.2) [1]	fmaxl(GLIBC_2.2) [1]	matherr(GLIBC_2.2) [2]	sqrtl(GLIBC_2.2) [1]
cargl(GLIBC_2.2) [1]	creall(GLIBC_2.2) [1]	fmin(GLIBC_2.2) [1]	modf(GLIBC_2.2) [1]	tan(GLIBC_2.2) [1]
casin(GLIBC_2.2) [1]	csin(GLIBC_2.2) [1]	fminf(GLIBC_2.2) [1]	modff(GLIBC_2.2) [1]	tanf(GLIBC_2.2) [1]
casinf(GLIBC_2.2) [1]	csinf(GLIBC_2.2) [1]	fminl(GLIBC_2.2) [1]	modfl(GLIBC_2.2) [1]	tanh(GLIBC_2.2) [1]
casinh(GLIBC_2.2) [1]	csinh(GLIBC_2.2) [1]	fmod(GLIBC_2.2) [1]	nan(GLIBC_2.2) [1]	tanhf(GLIBC_2.2) [1]
casinhf(GLIBC_2.2) [1]	csinhf(GLIBC_2.2) [1]	fmodf(GLIBC_2.2) [1]	nanf(GLIBC_2.2) [1]	tanhl(GLIBC_2.2) [1]
casinhl(GLIBC_2.2) [1]	csinhl(GLIBC_2.2) [1]	fmodl(GLIBC_2.2) [1]	nanl(GLIBC_2.2) [1]	tanl(GLIBC_2.2) [1]
casinl(GLIBC_2.2) [1]	csinl(GLIBC_2.2) [1]	frexp(GLIBC_2.2) [1]	nearbyint(GLIBC_2.2) [1]	tgamma(GLIBC_2.2) [1]
catan(GLIBC_2.2) [1]	csqrt(GLIBC_2.2) [1]	frexpf(GLIBC_2.2) [1]	nearbyintf(GLIBC_2.2) [1]	tgammaf(GLIBC_2.2) [1]
catanf(GLIBC_2.2) [1]	csqrtf(GLIBC_2.2) [1]	frexpl(GLIBC_2.2) [1]	nearbyintl(GLIBC_2.2) [1]	tgammal(GLIBC_2.2) [1]
catanh(GLIBC_2.2) [1]	csqrtl(GLIBC_2.2) [1]	gamma(GLIBC_2.2) [3]	nextafter(GLIBC_2.2) [1]	trunc(GLIBC_2.2) [1]
catanhf(GLIBC_2.2) [1]	ctan(GLIBC_2.2) [1]	gammaf(GLIBC_2.2) [2]	nextafterf(GLIBC_2.2) [1]	truncf(GLIBC_2.2) [1]
catanhl(GLIBC_2.2) [1]	ctanf(GLIBC_2.2) [1]	gammal(GLIBC_2.2) [2]	nextafterl(GLIBC_2.2) [1]	truncl(GLIBC_2.2) [1]
catanl(GLIBC_2.2) [1]	ctanh(GLIBC_2.2) [1]	hypot(GLIBC_2.2) [1]	nexttoward(GLIBC_2.2) [1]	y0(GLIBC_2.2) [1]
cbrt(GLIBC_2.2) [1]	ctanhf(GLIBC_2.2) [1]	hypotf(GLIBC_2.2) [1]	nexttowardf(GLIBC_2.2) [1]	y0f(GLIBC_2.2) [2]
cbrtf(GLIBC_2.2) [1]	ctanhl(GLIBC_2.2) [1]	hypotl(GLIBC_2.2) [1]	nexttowardl(GLIBC_2.2) [1]	y0l(GLIBC_2.2) [2]
cbrtl(GLIBC_2.2) [1]	ctanl(GLIBC_2.2) [1]	ilogb(GLIBC_2.2) [1]	pow(GLIBC_2.2) [1]	y1(GLIBC_2.2) [1]
ccos(GLIBC_2.2)	dremf(GLIBC_2.2)	ilogbf(GLIBC_2.2)	pow10(GLIBC_2.2)	y1f(GLIBC_2.2) [2]

[1]	[2]	[1]	[2]	
ccosf(GLIBC_2.2) [1]	dreml(GLIBC_2.2) [2]	ilogbl(GLIBC_2.2) [1]	pow10f(GLIBC_2.2) [2]	y1l(GLIBC_2.2) [2]
ccosh(GLIBC_2.2) [1]	erf(GLIBC_2.2) [1]	j0(GLIBC_2.2) [1]	pow10l(GLIBC_2.2) [2]	yn(GLIBC_2.2) [1]
ccoshf(GLIBC_2.2) [1]	erfc(GLIBC_2.2) [1]	j0f(GLIBC_2.2) [2]	powf(GLIBC_2.2) [1]	ynf(GLIBC_2.2) [2]
ccoshl(GLIBC_2.2) [1]	erfcf(GLIBC_2.2) [1]	j0l(GLIBC_2.2) [2]	powl(GLIBC_2.2) [1]	ynl(GLIBC_2.2) [2]
ccosl(GLIBC_2.2) [1]	erfcl(GLIBC_2.2) [1]	j1(GLIBC_2.2) [1]	remainder(GLIBC_2.2) [1]	
ceil(GLIBC_2.2) [1]	erff(GLIBC_2.2) [1]	j1f(GLIBC_2.2) [2]	remainderf(GLIBC_2.2) [1]	
ceilf(GLIBC_2.2) [1]	erfl(GLIBC_2.2) [1]	j1l(GLIBC_2.2) [2]	remainderl(GLIBC_2.2) [1]	
ceill(GLIBC_2.2) [1]	exp(GLIBC_2.2) [1]	jn(GLIBC_2.2) [1]	remquo(GLIBC_2.2) [1]	

Referenced Specification(s)

[1]. ISO POSIX (2003)

[2]. ISO C (1999)

[3]. SUSv2

An LSB conforming implementation shall provide the architecture specific data interfaces for Math specified in Table 1-30, with the full functionality as described in the referenced underlying specification.

Table 1-30. libm - Math Data Interfaces

signgam(GLIBC_2.2) [1]				

Referenced Specification(s)

[1]. ISO POSIX (2003)

1.5. Interfaces for libpthread

Table 1-31 defines the library name and shared object name for the libpthread library

Table 1-31. libpthread Definition

Library:	libpthread
SONAME:	libpthread.so.0

The behavior of the interfaces in this library is specified by the following specifications:

Large File Support
this specification
ISO POSIX (2003)

1.5.1. Realtime Threads

1.5.1.1. Interfaces for Realtime Threads

No external functions are defined for libpthread - Realtime Threads

1.5.2. Advanced Realtime Threads

1.5.2.1. Interfaces for Advanced Realtime Threads

No external functions are defined for libpthread - Advanced Realtime Threads

1.5.3. Posix Threads

1.5.3.1. Interfaces for Posix Threads

An LSB conforming implementation shall provide the architecture specific functions for Posix Threads specified in Table 1-32, with the full functionality as described in the referenced underlying specification.

Table 1-32. libpthread - Posix Threads Function Interfaces

_pthread_cleanup_pop(GLIBC_2.2) [1]	pthread_cancel(GLIBC_2.2) [2]	pthread_join(GLIBC_2.2) [2]	pthread_rwlock_destroy(GLIBC_2.2) [2]	pthread_setconcurrency(GLIBC_2.2) [2]
_pthread_cleanup_push(GLIBC_2.2) [1]	pthread_cond_broadcast(GLIBC_2.3.2) [2]	pthread_key_create(GLIBC_2.2) [2]	pthread_rwlock_init(GLIBC_2.2) [2]	pthread_setspecific(GLIBC_2.2) [2]
pread(GLIBC_2.2) [2]	pthread_cond_destroy(GLIBC_2.3.2) [2]	pthread_key_delete(GLIBC_2.2) [2]	pthread_rwlock_rdlock(GLIBC_2.2) [2]	pthread_sigmask(GLIBC_2.2) [2]
pread64(GLIBC_2.2) [3]	pthread_cond_init(GLIBC_2.3.2) [2]	pthread_kill(GLIBC_2.2) [2]	pthread_rwlock_timedrdlock(GLIBC_2.2) [2]	pthread_testcancel(GLIBC_2.2) [2]
pthread_attr_destroy(GLIBC_2.2) [2]	pthread_cond_signal(GLIBC_2.3.2) [2]	pthread_mutex_destroy(GLIBC_2.2) [2]	pthread_rwlock_timedwrlock(GLIBC_2.2) [2]	pwrite(GLIBC_2.2) [2]
pthread_attr_getdetachstate(GLIBC_2.2) [2]	pthread_cond_timedwait(GLIBC_2.3.2) [2]	pthread_mutex_init(GLIBC_2.2) [2]	pthread_rwlock_tryrdlock(GLIBC_2.2) [2]	pwrite64(GLIBC_2.2) [3]
pthread_attr_getgua	pthread_cond_wait(	pthread_mutex_lock	pthread_rwlock_try	sem_close(GLIBC_

rdsize(GLIBC_2.2) [2]	GLIBC_2.3.2) [2]	(GLIBC_2.2) [2]	wrlock(GLIBC_2.2) [2]	2.2) [2]
pthread_attr_getschedparam(GLIBC_2.2) [2]	pthread_condattr_destroy(GLIBC_2.2) [2]	pthread_mutex_trylock(GLIBC_2.2) [2]	pthread_rwlock_unlock(GLIBC_2.2) [2]	sem_destroy(GLIBC_2.2) [2]
pthread_attr_getstackaddr(GLIBC_2.2) [2]	pthread_condattr_getpshared(GLIBC_2.2) [2]	pthread_mutex_unlock(GLIBC_2.2) [2]	pthread_rwlock_wrlock(GLIBC_2.2) [2]	sem_getvalue(GLIBC_2.2) [2]
pthread_attr_getstacksize(GLIBC_2.2) [2]	pthread_condattr_init(GLIBC_2.2) [2]	pthread_mutexattr_destroy(GLIBC_2.2) [2]	pthread_rwlockattr_destroy(GLIBC_2.2) [2]	sem_init(GLIBC_2.2) [2]
pthread_attr_init(GLIBC_2.2) [2]	pthread_condattr_setpshared(GLIBC_2.2) [2]	pthread_mutexattr_getpshared(GLIBC_2.2) [2]	pthread_rwlockattr_getpshared(GLIBC_2.2) [2]	sem_open(GLIBC_2.2) [2]
pthread_attr_setdetachstate(GLIBC_2.2) [2]	pthread_create(GLIBC_2.2) [2]	pthread_mutexattr_gettype(GLIBC_2.2) [2]	pthread_rwlockattr_init(GLIBC_2.2) [2]	sem_post(GLIBC_2.2) [2]
pthread_attr_setguardsize(GLIBC_2.2) [2]	pthread_detach(GLIBC_2.2) [2]	pthread_mutexattr_init(GLIBC_2.2) [2]	pthread_rwlockattr_setpshared(GLIBC_2.2) [2]	sem_timedwait(GLIBC_2.2) [2]
pthread_attr_setschedparam(GLIBC_2.2) [2]	pthread_equal(GLIBC_2.2) [2]	pthread_mutexattr_setpshared(GLIBC_2.2) [2]	pthread_self(GLIBC_2.2) [2]	sem_trywait(GLIBC_2.2) [2]
pthread_attr_setstackaddr(GLIBC_2.2) [2]	pthread_exit(GLIBC_2.2) [2]	pthread_mutexattr_settype(GLIBC_2.2) [2]	pthread_setcancelstate(GLIBC_2.2) [2]	sem_unlink(GLIBC_2.2) [2]
pthread_attr_setstacksize(GLIBC_2.3.3) [2]	pthread_getspecific(GLIBC_2.2) [2]	pthread_once(GLIBC_2.2) [2]	pthread_setcanceltype(GLIBC_2.2) [2]	sem_wait(GLIBC_2.2) [2]

596

597 *Referenced Specification(s)*

598 **[1].** this specification

599 **[2].** ISO POSIX (2003)

600 **[3].** Large File Support

1.6. Interfaces for libgcc_s

601 Table 1-33 defines the library name and shared object name for the libgcc_s library

Table 1-33. libgcc_s Definition

Library:	libgcc_s
SONAME:	libgcc_s.so.1

The behavior of the interfaces in this library is specified by the following specifications:

this specification

1.6.1. Unwind Library

1.6.1.1. Interfaces for Unwind Library

An LSB conforming implementation shall provide the architecture specific functions for Unwind Library specified in Table 1-34, with the full functionality as described in the referenced underlying specification.

Table 1-34. libgcc_s - Unwind Library Function Interfaces

_Unwind_DeleteException(GCC_3.0) [1]	_Unwind_GetGR(GCC_3.0) [1]	_Unwind_GetLanguageSpecificData(GCC_3.0) [1]	_Unwind_RaiseException(GCC_3.0) [1]	_Unwind_SetGR(GCC_3.0) [1]
_Unwind_ForcedUnwind(GCC_3.0) [1]	_Unwind_GetIP(GCC_3.0) [1]	_Unwind_GetRegionStart(GCC_3.0) [1]	_Unwind_Resume(GCC_3.0) [1]	_Unwind_SetIP(GCC_3.0) [1]

Referenced Specification(s)

[1]. this specification

1.7. Interface Definitions for libgcc_s

The following interfaces are included in libgcc_s and are defined by this specification. Unless otherwise noted, these interfaces shall be included in the source standard.

Other interfaces listed above for libgcc_s shall behave as described in the referenced base document.

_Unwind_DeleteException

Name

`_Unwind_DeleteException` — private C++ error handling method

Synopsis

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

Description

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

_Unwind_ForcedUnwind

Name

`_Unwind_ForcedUnwind` — private C++ error handling method

Synopsis

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

Description

`_Unwind_ForcedUnwind` raises an exception for forced unwinding, passing along the given exception *`object`*, which should have its *exception_class* and *exception_cleanup* fields set. The exception *`object`* has been allocated by the language-specific runtime, and has a language-specific format, except that it shall contain an _Unwind_Exception struct.

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

Return Value

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

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

_URC_NO_REASON

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

_URC_END_OF_STACK

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

_URC_FATAL_PHASE2_ERROR

: The `stop` function may return this code for other fatal conditions like stack corruption.

_Unwind_GetGR

Name

`_Unwind_GetGR` — private C++ error handling method

Synopsis

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

Description

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

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

_Unwind_GetIP

Name

`_Unwind_GetIP` — private C++ error handling method

Synopsis

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

Description

`_Unwind_GetIP` returns the instruction pointer value for the routine identified by the unwind *`context`*.

_Unwind_GetLanguageSpecificData

Name

`_Unwind_GetLanguageSpecificData` — private C++ error handling method

Synopsis

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

Description

`_Unwind_GetLanguageSpecificData` returns the address of the language specific data area for the current stack frame.

_Unwind_GetRegionStart

Name

`_Unwind_GetRegionStart` — private C++ error handling method

Synopsis

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

Description

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

_Unwind_RaiseException

Name

`_Unwind_RaiseException` — private C++ error handling method

Synopsis

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

Description

`_Unwind_RaiseException` raises an exception, passing along the given exception *`object`*, which should have its *`exception_class`* and *`exception_cleanup`* fields set. The exception object has been allocated by the language-specific runtime, and has a language-specific format, exception that it shall contain an `_Unwind_Exception`.

Return Value

`_Unwind_RaiseException` does not return unless an error condition is found. If an error condition occurs, an `_Unwind_Reason_Code` is returnd:

_URC_END_OF_STACK

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

_URC_FATAL_PHASE1_ERROR

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

_URC_FATAL_PHASE2_ERROR

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

_Unwind_Resume

Name

`_Unwind_Resume` — private C++ error handling method

Synopsis

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

Description

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

_Unwind_SetGR

Name

`_Unwind_SetGR` — private C++ error handling method

Synopsis

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

Description

`_Unwind_SetGR` sets the *`value`* of the register *`index`*ed for the routine identified by the unwind *`context`*.

_Unwind_SetIP

Name

`_Unwind_SetIP` — private C++ error handling method

Synopsis

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

Description

`_Unwind_SetIP` sets the *`value`* of the instruction pointer for the routine identified by the unwind *`context`*

1.8. Interfaces for libdl

Table 1-35 defines the library name and shared object name for the libdl library

Table 1-35. libdl Definition

Library:	libdl
SONAME:	libdl.so.2

The behavior of the interfaces in this library is specified by the following specifications:

this specification
ISO POSIX (2003)

1.8.1. Dynamic Loader

1.8.1.1. Interfaces for Dynamic Loader

An LSB conforming implementation shall provide the architecture specific functions for Dynamic Loader specified in Table 1-36, with the full functionality as described in the referenced underlying specification.

Table 1-36. libdl - Dynamic Loader Function Interfaces

dladdr(GLIBC_2.0) [1]	dlclose(GLIBC_2.0) [2]	dlerror(GLIBC_2.0) [2]	dlopen(GLIBC_2.1) [1]	dlsym(GLIBC_2.0) [1]

Referenced Specification(s)

[1]. this specification

[2]. ISO POSIX (2003)

1.9. Interfaces for libcrypt

Table 1-37 defines the library name and shared object name for the libcrypt library

Table 1-37. libcrypt Definition

Library:	libcrypt
SONAME:	libcrypt.so.1

The behavior of the interfaces in this library is specified by the following specifications:

ISO POSIX (2003)

1.9.1. Encryption

1.9.1.1. Interfaces for Encryption

An LSB conforming implementation shall provide the architecture specific functions for Encryption specified in Table 1-38, with the full functionality as described in the referenced underlying specification.

Table 1-38. libcrypt - Encryption Function Interfaces

crypt(GLIBC_2.0)	encrypt(GLIBC_2.0	setkey(GLIBC_2.0)		

[1]	) [1]	[1]		

Referenced Specification(s)

[1]. ISO POSIX (2003)

II. Utility Libraries

Chapter 2. Libraries

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

2.1. Interfaces for libz

Table 2-1 defines the library name and shared object name for the libz library

Table 2-1. libz Definition

Library:	libz
SONAME:	libz.so.1

2.1.1. Compression Library

2.1.1.1. Interfaces for Compression Library

No external functions are defined for libz - Compression Library

2.2. Interfaces for libncurses

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

Table 2-2. libncurses Definition

Library:	libncurses
SONAME:	libncurses.so.5

2.2.1. Curses

2.2.1.1. Interfaces for Curses

No external functions are defined for libncurses - Curses

2.3. Interfaces for libutil

Table 2-3 defines the library name and shared object name for the libutil library

Table 2-3. libutil Definition

Library:	libutil
SONAME:	libutil.so.1

The behavior of the interfaces in this library is specified by the following specifications:

this specification

2.3.1. Utility Functions

2.3.1.1. Interfaces for Utility Functions

An LSB conforming implementation shall provide the architecture specific functions for Utility Functions specified in Table 2-4, with the full functionality as described in the referenced underlying specification.

Table 2-4. libutil - Utility Functions Function Interfaces

forkpty(GLIBC_2.0) [1]	login_tty(GLIBC_2.0) [1]	logwtmp(GLIBC_2.0) [1]		
login(GLIBC_2.0) [1]	logout(GLIBC_2.0) [1]	openpty(GLIBC_2.0) [1]		

Referenced Specification(s)

[1]. this specification

Appendix A. Alphabetical Listing of Interfaces

A.1. libgcc_s

The behaviour of the interfaces in this library is specified by the following Standards.

this specification

Table A-1. libgcc_s Function Interfaces

_Unwind_DeleteException[1]	_Unwind_GetLanguageSpecificData[1]	_Unwind_SetGR[1]
_Unwind_ForcedUnwind[1]	_Unwind_GetRegionStart[1]	_Unwind_SetIP[1]
_Unwind_GetGR[1]	_Unwind_RaiseException[1]	
_Unwind_GetIP[1]	_Unwind_Resume[1]	

Linux Packaging Specification

Linux Packaging Specification

Table of Contents

I. Package Format and Installation 48

1. Software Installation 1
    1.1. Package Dependencies 1
    1.2. Package Architecture Considerations 1

I. Package Format and Installation

Chapter 1. Software Installation

1.1. Package Dependencies

The LSB runtime environment shall provde the following dependencies.

lsb-core-ia64

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

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

1.2. Package Architecture Considerations

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

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

Free Documentation License

Free Documentation License

Table of Contents

A. GNU Free Documentation License 1
A.1. PREAMBLE 1
A.2. APPLICABILITY AND DEFINITIONS 1
A.3. VERBATIM COPYING 2
A.4. COPYING IN QUANTITY 2
A.5. MODIFICATIONS 3
A.6. COMBINING DOCUMENTS 4
A.7. COLLECTIONS OF DOCUMENTS 4
A.8. AGGREGATION WITH INDEPENDENT WORKS 4
A.9. TRANSLATION 5
A.10. TERMINATION 5
A.11. FUTURE REVISIONS OF THIS LICENSE 5
A.12. How to use this License for your documents 5

Appendix A. GNU Free Documentation License

Version 1.1, March 2000

A.1. PREAMBLE

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

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

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

A.2. APPLICABILITY AND DEFINITIONS

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

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

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

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

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

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

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

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

A.3. VERBATIM COPYING

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

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

A.4. COPYING IN QUANTITY

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

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

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

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

A.5. MODIFICATIONS

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

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

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

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

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

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

A.6. COMBINING DOCUMENTS

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

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

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

A.7. COLLECTIONS OF DOCUMENTS

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

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

A.8. AGGREGATION WITH INDEPENDENT WORKS

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

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

A.9. TRANSLATION

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

A.10. TERMINATION

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

A.11. FUTURE REVISIONS OF THIS LICENSE

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

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

A.12. How to use this License for your documents

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

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

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

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