BS ISO/IEC 23360-1-2:2021

BSI Standards Publication

Linux Standard Base (LSB)

Part 1-2: Core specification generic part

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23360-1-2:2021

National foreword

This British Standard is the UK implementation of ISO/IEC 23360-1-2:2021.

The UK participation in its preparation was entrusted to Technical Committee IST/5, Programming languages, their environments and system software interfaces.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2021
Published by BSI Standards Limited 2021

ISBN 978 0 539 04906 0

ICS 35.080

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 October 2021.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23360-1-2:2021

INTERNATIONAL STANDARD

ISO/IEC 23360-1-2

First edition
2021-10

Linux Standard Base (LSB) —

Part 1-2:
Core specification generic part

Reference number
ISO/IEC 23360-1-2:2021(E)

Documents
Document Sharing Hub
© ISO/IEC 2021

Full document details: BS ISO/IEC 23360-1-2:2021

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by the Linux Foundation as Linux Standard Base (LSB): Core specification generic part and drafted in accordance with its editorial rules. It was assigned to Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*, and adopted by National Bodies.

This first edition of ISO/IEC 23360-1-2 cancels and replaces ISO/IEC 23360-1:2006, which has been technically revised.

This document is based on “The GNU Free Documentation License, version 1.1”. The license is available at https://www.gnu.org/licenses/old-licenses/fdl-1.1.html.

A list of all parts in the ISO/IEC 23660 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user’s national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Contents

Foreword ... **iii**
Introduction ... **vii**
I Introductory Elements ... **1**
1 Scope ... 2
2 References ... 3
2.1 Normative References ... 3
2.2 Informative References/Bibliography ... 5
3 Requirements ... 8
3.1 Relevant Libraries ... 8
3.2 LSB Implementation Conformance ... 8
3.3 LSB Application Conformance ... 10
4 Terms and Definitions ... 11
5 Documentation Conventions ... 13
6 Relationship To ISO/IEC 9945 POSIX ... 14
7 Relationship To Other Linux Foundation Specifications ... 15
II Executable And Linking Format (ELF) ... **16**
8 Introduction ... 17
9 Low Level System Information ... 18
9.1 Operating System Interface ... 18
9.2 Machine Interface ... 18
10 Object Format ... 19
10.1 Object Files ... 19
10.2 Sections ... 19
10.3 Special Sections ... 23
10.4 Symbol Mapping ... 29
10.5 DWARF Extensions ... 29
10.6 Exception Frames ... 31
10.7 Symbol Versioning ... 36
10.8 ABI note tag ... 40
11 Dynamic Linking ... 41
11.1 Program Loading and Dynamic Linking ... 41
11.2 Program Header ... 41
11.3 Dynamic Entries ... 41
12 C++ Class Representations ... 46
12.1 C++ Data Representation ... 46
13 Symbol Mapping ... 50
13.1 Symbol Mapping ... 50
III Base Libraries ... **51**
14 Base Libraries ... 52
14.1 Introduction ... 52
14.2 Program Interpreter ... 52
14.3 Interfaces for libc ... 52
14.4 Data Definitions for libc ... 73
14.5 Interface Definitions for libc ... 180
14.6 Interfaces for libm ... 393
14.7 Data Definitions for libm ... 397
14.8 Interface Definitions for libm ... 403
14.9 Interfaces for libpthread ... 428
14.10 Data Definitions for libpthread ... 434
14.11 Interface Definitions for libpthread ... 439

AVA Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

14.12 Interfaces for libgcc_s 444
14.13 Data Definitions for libgcc_s 445
14.14 Interface Definitions for libgcc_s 446
14.15 Interfaces for libdl 452
14.16 Data Definitions for libdl 453
14.17 Interface Definitions for libdl 454
14.18 Interfaces for librt 457
14.19 Data Definitions for librt 459
14.20 Interfaces for libcrypt 461
14.21 Data Definitions for libcrypt 462
14.22 Interface Definitions for libcrypt 462
14.23 Interfaces for libpam 464
14.24 Data Definitions for libpam 465
14.25 Interface Definitions for libpam 467
IV Utility Libraries 481
15 Utility Libraries 482
15.1 Introduction 482
15.2 Interfaces for libz 482
15.3 Data Definitions for libz 483
15.4 Interface Definitions for libz 486
15.5 Interfaces for libncurses 533
15.6 Data Definitions for libncurses 538
15.7 Interface Definitions for libncurses 546
15.8 Interfaces for libncursesw 554
15.9 Data Definitions for libncursesw 560
15.10 Interface Definitions for libncursesw 587
15.11 Interfaces for libutil 587
15.12 Data Definitions for libutil 588
15.13 Interface Definitions for libutil 588
V C++ Libraries 594
16 Libraries 595
16.1 Interfaces for libstdcxx 595
16.2 Interface Definitions for libstdcxx 850
VI Commands and Utilities 851
17 Commands and Utilities 852
17.1 Commands and Utilities 852
17.2 Command Behavior 853
VII Execution Environment 917
18 File System Hierarchy 918
18.1 /dev: Device Files 918
18.2 /etc: Host-specific system configuration 918
18.3 User Accounting Databases 920
18.4 Path For System Administration Utilities 920
19 Additional Recommendations 921
19.1 Recommendations for applications on ownership and permissions 921
20 Additional Behaviors 923
20.1 Mandatory Optional Behaviors 923
20.2 Optional Mandatory Behaviors 924
20.3 Executable Scripts 924
21 Localization 926
21.1 Introduction 926

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

21.2 Regular Expressions ... 926
21.3 Pattern Matching Notation ... 926
VIII System Initialization ... 928
22 System Initialization ... 929
22.1 Cron Jobs ... 929
22.2 Init Script Actions ... 930
22.3 Comment Conventions for Init Scripts ... 932
22.4 Installation and Removal of Init Scripts ... 934
22.5 Run Levels ... 935
22.6 Facility Names ... 935
22.7 Script Names ... 936
22.8 Init Script Functions ... 936
IX Users & Groups ... 939
23 Users & Groups ... 940
23.1 User and Group Database ... 940
23.2 User & Group Names ... 940
23.3 User ID Ranges ... 941
23.4 Rationale ... 941
X Network Security Services ... 942
24 Libraries ... 943
24.1 Interfaces for libnspr4 ... 943
24.2 Data Definitions for libnspr4 ... 945
24.3 Interfaces for libnss3 ... 955
24.4 Data Definitions for libnss3 ... 956
24.5 Interfaces for libssl3 ... 979
24.6 Data Definitions for libssl3 ... 980
XI Package Format and Installation ... 990
25 Software Installation ... 991
25.1 Introduction ... 991
25.2 Package File Format ... 991
25.3 Package Script Restrictions ... 1011
25.4 Package Tools ... 1011
25.5 Package Naming Conventions ... 1011
25.6 Package Dependencies ... 1012
25.7 Package Architecture Considerations ... 1013
Annex A Alphabetical Listing of Interfaces by Library ... 1014
A.1 libc ... 1014
A.2 libcrypt ... 1029
A.3 libdl ... 1029
A.4 libgcc_s ... 1029
A.5 libm ... 1029
A.6 libncurses ... 1033
A.7 libncursesw ... 1036
A.8 libpam ... 1043
A.9 libpthread ... 1043
A.10 librt ... 1047
A.11 libutil ... 1047
A.12 libz ... 1048
A.13 libnspr4 ... 1049
A.14 libnss3 ... 1050
A.15 libssl3 ... 1051

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

Introduction

The LSB defines a binary interface for application programs that are compiled and packaged for LSB-conforming implementations on many different hardware architectures. A binary specification must include information specific to the computer processor architecture for which it is intended. To avoid the complexity of conditional descriptions, the specification has instead been divided into generic parts which are augmented by one of several architecture-specific parts, depending on the target processor architecture; the generic part will indicate when reference must be made to the architecture part, and vice versa.

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.

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

1. The first number (`x`) is the major version number. Versions sharing the same major version number shall be compatible in a backwards direction; that is, a newer version shall be compatible with an older version. Any deletion of a library results in a new major version number. Interfaces marked as deprecated may be removed from the specification at a major version change.
2. The second number (`y`) is the minor version number. Libraries and individual interfaces may be added, but not removed. Interfaces may be marked as deprecated at a minor version change. Other minor changes may be permitted at the discretion of the LSB workgroup.
3. The third number (`z`), if present, is the editorial level. Only editorial changes should be included in such versions.

Since this specification is a descriptive Application Binary Interface, and not a source level API specification, it is not possible to make a guarantee of 100% backward compatibility between major releases. However, it is the intent that those parts of the binary interface that are visible in the source level API will remain backward compatible from version to version, except where a feature marked as "Deprecated" in one release may be removed from a future release. Implementors are strongly encouraged to make use of symbol versioning to permit simultaneous support of applications conforming to different releases of this specification.

AVA Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

LSB is a trademark of the Linux Foundation. Developers of applications or implementations interested in using the trademark should see the Linux Foundation Certification Policy for details.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23360-1-2:2021

I Introductory Elements

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23360-1-2:2021

1 Scope

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 part describing those parts of the interface that remain constant across all implementations of the LSB, and an architecture-specific part describing the parts of the interface that vary by processor architecture. Together, the common part and the relevant architecture-specific part for a single hardware architecture provide a complete interface specification for compiled application programs on systems that share a common hardware architecture.

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 provides 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.

This is the common part of the Core module of the Linux Standard Base (LSB), LSB Core - Generic. This module provides the fundamental system interfaces, libraries, and runtime environment upon which all conforming applications and libraries depend.

LSB Core - Generic, the common part, should be used in conjunction with an architecture-specific part. Whenever a section of the common part is supplemented by architecture-specific information, the common part includes a reference to the architecture-specific part. Architecture-specific parts of the LSB Core Specification may also contain additional information that is not referenced in the common part.

Interfaces described in this part of the LSB Core Specification are mandatory except where explicitly listed otherwise. Interfaces described in the LSB Core module are supplemented by other LSB modules. All other modules depend on the presence of LSB Core.

AVA Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

2 References

2.1 Normative References

The following specifications are incorporated by reference into this specification. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced specification (including any amendments) applies.

Note: Where copies of a referenced specification are available on the World Wide Web, a Uniform Resource Locator (URL) is given, for informative purposes only. Such URL might at any given time resolve to a more recent copy of the specification, or be out of date (not resolve). Reference copies of specifications at the revision level indicated may be found at the Linux Foundation's Reference Specifications (http://refspecs.linuxbase.org) site.

Table 2-1 Normative References

Name	Title	URL
Filesystem Hierarchy Standard	Filesystem Hierarchy Standard (FHS) 3.0	http://refspecs.linuxbase.org/fhs
ISO C (1999)	ISO/IEC 9899:1999 - Programming Languages -- C	
ISO/IEC 14882: 2003 C++ Language	ISO/IEC 14882: 2003 Programming languages --C++	
Itanium™ C++ ABI	Itanium™ C++ ABI (Revision 1.86)	http://refspecs.linuxfoundation.org/cxxabi-1.86.html
Large File Support	Large File Support	http://www.UNIX-systems.org/version2/whatsnew/lfs20mar.html
Libncursesw API	Libncursesw API	http://invisible-island.net/ncurses/man/ncurses.3x.html
Libncursesw Placeholder	Libncursesw Specification Placeholder	http://refspecs.linux-foundation.org/libncursesw/libncurses.html
POSIX 1003.1-2001 (ISO/IEC 9945-2003)	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	http://www.unix.org/version3/

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Name	Title	URL
	(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 Including Technical Cor. 1: 2004	
POSIX 1003.1-2008 (ISO/IEC 9945-2009)	Portable Operating System Interface (POSIX®) 2008 Edition / The Open Group Technical Standard Base Specifications, Issue 7	http://www.unix.org/version4/
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
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	http://refspecs.linuxfoundation.org/svid4/
System V ABI	System V Application Binary Interface, Edition 4.1	http://www.sco.com/developers/devspecs/gabi41.pdf
System V ABI Update	System V Application Binary Interface -	http://www.sco.com/developers/gabi/2003-12-17/contents.html

AVA Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

Name	Title	URL
	DRAFT - 17 December 2003	
X/Open Curses, Issue 7	X/Open Curses, Issue 7 (ISBN: 1-931624-83-6, The Open Group, November 2009)	https://www2.opengroup.org/ogsys/catalog/C094

2.2 Informative References/Bibliography

The documents listed below provide essential background information to implementors of this specification. These references are included for information only, and do not represent normative parts of this specification.

Table 2-2 Other References

Name	Title	URL
DWARF Debugging Information Format, Version 4	DWARF Debugging Information Format, Version 4 (June 10, 2010)	http://www.dwarfstd.org/doc/DWARF4.pdf
IEC 60559/IEEE 754 Floating Point	IEC 60559:1989 Binary floating-point arithmetic for microprocessor systems	http://www.ieee.org/
ISO/IEC TR14652	ISO/IEC Technical Report 14652:2002 Specification method for cultural conventions	
ITU-T V.42	International Telecommunication Union Recommendation V.42 (2002): Error-correcting procedures for DCEs using asynchronous-to-synchronous conversionITUV	http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-V.42
Li18nux Globalization Specification	LI18NUX 2000 Globalization Specification, Version 1.0 with Amendment 4	http://www.openi18n.org/docs/html/LI18NUX-2000-amd4.htm
Linux Allocated Device Registry	LINUX ALLOCATED DEVICES	http://www.lanana.org/docs/device-list/devices-2.6+.txt

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Name	Title	URL
Linux Assigned Names And Numbers Authority	Linux Assigned Names And Numbers Authority	http://www.lanana.org/
Mozilla's NSS SSL Reference	Mozilla's NSS SSL Reference	http://www.mozilla.org/projects/security/pki/nss/ref/ssl/
NSPR Reference	Mozilla's NSPR Reference	http://refspecs.linuxfoundation.org/NSPR_API_Reference/NSPR_API.html
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 1950: ZLIB Compressed Data Format Specication	IETF RFC 1950: ZLIB Compressed Data Format Specification	http://www.ietf.org/rfc/rfc1950.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
RFC 2821:Simple Mail Transfer Protocol	IETF RFC 2821: Simple Mail Transfer Protocol	http://www.ietf.org/rfc/rfc2821.txt
RFC 2822:Internet Message Format	IETF RFC 2822: Internet Message Format	http://www.ietf.org/rfc/rfc2822.txt
RFC 5531/4506 RPC & XDR	IETF RFC 5531 & 4506	http://www.ietf.org/

AVA Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

Name	Title	URL
RFC 791:Internet Protocol	IETF RFC 791: Internet Protocol Specification	http://www.ietf.org/rfc/rfc791.txt
RPM Package Format	RPM Package Format V3.0	http://www.rpm.org/max-rpm/s1-rpm-file-format-rpm-file-format.html
zlib Manual	zlib 1.2 Manual	http://www.gzip.org/zlib/

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

3 Requirements

3.1 Relevant Libraries

The libraries listed in Table 3-1 shall be available on a Linux Standard Base system, with the specified runtime names. The libraries listed in Table 3-2 are architecture specific, but shall be available on all LSB conforming systems. This list may be supplemented or amended by the relevant architecture specific part of the LSB Core Specification.

Table 3-1 Standard Library Names

Library	Runtime Name
libcrypt	libcrypt.so.1
libdl	libdl.so.2
libgcc_s	libgcc_s.so.1
libncurses	libncurses.so.5
libncursesw	libncursesw.so.5
libnspr4	libnspr4.so
libnss3	libnss3.so
libpam	libpam.so.0
libpthread	libpthread.so.0
librt	librt.so.1
libssl3	libssl3.so
libstdcxx	libstdc++.so.6
libutil	libutil.so.1
libz	libz.so.1

Table 3-2 Standard Library Names defined in the Architecture Specific Parts of the LSB Core Specification

Library	Runtime Name
libc	See architecture specific part.
libm	See architecture specific part.
proginterp	See architecture specific part.

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 is necessarily architecture specific, and must provide the interfaces specified by both the generic LSB Core specification (LSB

AVA Documents Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

Core - Generic) and the relevant architecture specific part of the LSB Core Specification.

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

A conforming implementation shall satisfy the following requirements:

- A processor architecture represents a family of related processors which may not have identical feature sets. The architecture specific parts of the LSB Core Specification that supplement this specification for a given target processor architecture describe a minimum acceptable processor. The implementation shall provide all features of this processor, whether in hardware or through emulation transparent to the application.
- The implementation shall be capable of executing compiled applications having the format and using the system interfaces described in this specification.
- The implementation shall provide libraries containing the interfaces specified by this specification, 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 specification.
- The map of virtual memory provided by the implementation shall conform to the requirements of this specification.
- 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 specification.
- 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 specification in the format defined here and in other documents normatively included by reference. All commands and utilities shall behave as required by this specification. The implementation shall also provide all mandatory components of an application's runtime environment that are included or referenced in this specification.
- 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.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

3.3 LSB Application Conformance

A conforming application containing object files is necessarily architecture specific, and must conform to both the generic LSB Core specification (LSB Core - Generic) and the relevant architecture specific part of the LSB Core Specification. A conforming application which contains no object files may be architecture neutral. Architecture neutral applications shall conform only to the requirements of the generic LSB Core specification (LSB Core - Generic).

A conforming application shall satisfy the following requirements:

- Executable files shall be either object files in the format defined in the Object Format section of this specification, or script files in a scripting language where the interpreter is required by this specification.
- Object files shall participate in dynamic linking as defined in the Program Loading and Linking section of this specification.
- Object files shall employ only the instructions, traps, and other low-level facilities defined as being for use by applications in the Low-Level System Information section of this specification
- If the application requires any optional interface defined in this specification in order to be installed or to execute successfully, the requirement for that optional interface shall be stated in the application's documentation.
- The application shall not use any interface or data format that is not required to be provided by a conforming implementation, unless such an interface or data format is supplied by another application through direct invocation of that application during execution. The other application must also be a conforming application, and the use of such interface or data format, as well as its source (in other words, the other conforming application), shall be identified in the documentation of the application.
- The application shall not use any values for a named interface that are reserved for vendor extensions.

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

AVA Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

4 Terms and Definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 2382, ISO 80000–2, and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

4.1

archLSB

Some LSB specification documents have both a generic, architecture-neutral part and an architecture-specific part. The latter describes elements whose definitions may be unique to a particular processor architecture. The term archLSB may be used in the generic part to refer to the corresponding section of the architecture-specific part.

4.2

Binary Standard, ABI

The total set of interfaces that are available to be used in the compiled binary code of a conforming application, including the run-time details such as calling conventions, binary format, C++ name mangling, etc.

4.3

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.

4.4

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.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

4.5

Source Standard, API

The total set of interfaces that are available to be used in the source code of a conforming application. Due to translations, the Binary Standard and the Source Standard may contain some different interfaces.

4.6

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.

4.7

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.

In addition, for the portions of this specification which build on IEEE Std 1003.1-2001, the definitions given in *IEEE Std 1003.1-2001, Base Definitions, Chapter 3* apply.

AVA Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

5 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) [SUSv4]

refers to the interface named `forkpty()` with symbol version `GLIBC_2.0` that is defined in the reference indicated by the tag `SUSv4`.

> **Note:** For symbols with versions which differ between architectures, the symbol versions are defined in the architecture specific parts of of this module specification only. In the generic part, they will appear without symbol versions.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

6 Relationship To ISO/IEC 9945 POSIX

This specification includes many interfaces described in POSIX 1003.1-2008 (ISO/IEC 9945-2009). Unless otherwise specified, such interfaces should behave exactly as described in that specification. Any conflict between the requirements described here and the POSIX 1003.1-2008 (ISO/IEC 9945-2009) standard is unintentional, except as explicitly noted otherwise.

> **Note:** In addition to the differences noted in this specification, a report, *ISO/IEC TR 24715-Technical Report on the Conflicts Between the ISO/IEC 9945 (POSIX) Standard and the Linux Standard Base Specification (LSB)*, identifies the differences between edition 3.1 of this specification and *POSIX 1003.1-2001 (ISO/IEC 9945-2003)* (more precisely, POSIX 2001 plus the first two corrigenda, informally known as the 2004 edition). It is the long term plan of the Linux Foundation to converge the LSB Core specification with the ISO/IEC POSIX specification.

The LSB Specification Authority is responsible for deciding the meaning of conformance to normative referenced standards in the LSB context. Problem reports regarding underlying or referenced standards in any other context will be referred to the relevant maintenance body for that standard.

AVA Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

7 Relationship To Other Linux Foundation Specifications

The LSB is the base for several other specification projects under the umbrella of the Linux Foundation (LF). This specification is the foundation, and other specifications build on the interfaces defined here. However, beyond those specifications listed as Normative References, this specification has no dependencies on other LF projects.

ISO/IEC 23360 corresponds to an earlier edition of this specification (version 3.1), published as an ISO/IEC standard in 2006 after submission by the Linux Foundation. The ISO edition is also the subject of the technical report ISO/IEC TR 24715 referenced in the previous chapter.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

II Executable And Linking Format (ELF)

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

8 Introduction

Executable and Linking Format (ELF) defines the object format for compiled applications. This specification supplements the information found in System V ABI Update and is intended to document additions made since the publication of that document.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

9 Low Level System Information

9.1 Operating System Interface

LSB-conforming applications shall assume that stack, heap and other allocated memory regions will be non-executable. The application must take steps to make them executable if needed.

9.2 Machine Interface

9.2.1 Data Representation

LSB-conforming applications shall use the data representation as defined in the Arcitecture specific ELF documents.

9.2.1.1 Fundamental Types

In addition to the fundamental types specified in the relevant architecture specific part of the LSB Core Specification, a 1 byte data type is defined here.

Table 9-1 Scalar Types

Type	C	C++	`sizeof`	Align-ment (bytes)	Architec-ture Rep-resenta-tion
Integral	_Bool	bool	1	1	byte

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

10 Object Format

10.1 Object Files

LSB-conforming implementations shall support the Executable and Linking Format (ELF) object file format as defined by the following documents:

- System V ABI
- System V ABI Update
- the relevant architecture specific ABI supplement.
- this specification
- the relevant architecture specific part of the LSB Core Specification

Conforming implementations may also support other unspecified object file formats.

10.2 Sections

10.2.1 Introduction

As described in System V ABI, an ELF object file contains a number of *sections*.

10.2.2 Sections Types

The section header table is an array of `Elf32_Shdr` or `Elf64_Shdr` structures as described in System V ABI. The *`sh_type`* member shall be either a value from Table 10-1, drawn from the System V ABI, or one of the additional values specified in Table 10-2.

A section header's *`sh_type`* member specifies the sections's semantics.

10.2.2.1 ELF Section Types

The following section types are defined in the System V ABI and the System V ABI Update.

Table 10-1 ELF Section Types

Name	Value	Description
SHT_DYNAMIC	0x6	The section holds information for dynamic linking. Currently, an object file shall have only one dynamic section, but this restriction may be relaxed in the future. See `Dynamic Section' in Chapter 5 of System V ABI Update for details.
SHT_DYNSYM	0xb	This section holds a minimal set of symbols adequate for dynamic

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Name	Value	Description
		linking. See also SHT_SYMTAB. Currently, an object file may have either a section of SHT_SYMTAB type or a section of SHT_DYNSYM type, but not both. This restriction may be relaxed in the future.
SHT_FINI_ARRAY	0xf	This section contains an array of pointers to termination functions, as described in `Initialization and Termination Functions' in Chapter 5 of System V ABI Update. Each pointer in the array is taken as a parameterless procedure with a void return.
SHT_HASH	0x5	The section holds a symbol hash table. Currently, an object file shall have only one hash table, but this restriction may be relaxed in the future. See `Hash Table' in Chapter 5 of System V ABI Update for details.
SHT_INIT_ARRAY	0xe	This section contains an array of pointers to initialization functions, as described in `Initialization and Termination Functions' in Chapter 5 of System V ABI Update. Each pointer in the array is taken as a parameterless procedure with a void return.
SHT_NOBITS	0x8	A section of this type occupies no space in the file but otherwise re-

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

Name	Value	Description
		sembles SHT_PROG-BITS. Although this section contains no bytes, the sh_offset member contains the conceptual file offset.
SHT_NOTE	0x7	The section holds information that marks the file in some way. See `Note Section' in Chapter 5 of System V ABI Update for details.
SHT_NULL	0x0	This value marks the section header as inactive; it does not have an associated section. Other members of the section header have undefined values.
SHT_PREINIT_ARRAY	0x10	This section contains an array of pointers to functions that are invoked before all other initialization functions, as described in `Initialization and Termination Functions' in Chapter 5 of System V ABI Update. Each pointer in the array is taken as a parameterless proceure with a void return.
SHT_PROGBITS	0x1	The section holds information defined by the program, whose format and meaning are determined solely by the program.
SHT_REL	0x9	The section holds relocation entries without explicit addends, such as type Elf32_Rel for the 32-bit class of object files or type Elf64_Rel for the 64-bit class of object files. An object

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Name	Value	Description
		file may have multiple relocation sections. See `Relocation' in Chapter 4 of System V ABI Update for details.
SHT_RELA	0x4	The section holds relocation entries with explicit addends, such as type Elf32_Rela for the 32-bit class of object files or type Elf64_Rela for the 64-bit class of object files. An object file may have multiple relocation sections. See `Relocation' in Chapter 4 of System V ABI Update for details.
SHT_STRTAB	0x3	The section holds a string table. An object file may have multiple string table sections. See `String Table' in Chapter 4 of System V ABI Update for details.
SHT_SYMTAB	0x2	This section holds a symbol table. Currently, an object file may have either a section of SHT_SYMTAB type or a section of SHT_DYNSYM type, but not both. This restriction may be relaxed in the future. Typically, SHT_SYMTAB provides symbols for link editing, though it may also be used for dynamic linking. As a complete symbol table, it may contain many symbols unnecessary for dynamic linking.

10.2.2.2 Additional Section Types

The following additional section types are defined here.

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

Table 10-2 Additional Section Types

Name	Value	Description
SHT_GNU_verdef	0x6ffffffd	This section contains the symbol versions that are provided.
SHT_GNU_verneed	0x6ffffffe	This section contains the symbol versions that are required.
SHT_GNU_versym	0x6fffffff	This section contains the Symbol Version Table.

10.3 Special Sections

10.3.1 Special Sections

Various sections hold program and control information. Sections in the lists below are used by the system and have the indicated types and attributes.

10.3.1.1 ELF Special Sections

The following sections are defined in the System V ABI and the System V ABI Update.

Table 10-3 ELF Special Sections

Name	Type	Attributes
.bss	SHT_NOBITS	SHF_ALLOC+SHF_WRITE
.comment	SHT_PROGBITS	SHF_MERGE+SHF_STRINGS
.data	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE
.data1	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE
.debug	SHT_PROGBITS	0
.dynamic	SHT_DYNAMIC	SHF_ALLOC+SHF_WRITE
.dynstr	SHT_STRTAB	SHF_ALLOC
.dynsym	SHT_DYNSYM	SHF_ALLOC
.fini	SHT_PROGBITS	SHF_ALLOC+SHF_EXECINSTR
.fini_array	SHT_FINI_ARRAY	SHF_ALLOC+SHF_WRITE

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Name	Type	Attributes
.hash	SHT_HASH	SHF_ALLOC
.init	SHT_PROGBITS	SHF_ALLOC+SHF_EX-ECINSTR
.init_array	SHT_INIT_ARRAY	SHF_AL-LOC+SHF_WRITE
.interp	SHT_PROGBITS	SHF_ALLOC
.line	SHT_PROGBITS	0
.note	SHT_NOTE	0
.preinit_array	SHT_PREINIT_ARRAY	SHF_AL-LOC+SHF_WRITE
.rodata	SHT_PROGBITS	SHF_AL-LOC+SHF_MERGE+SH F_STRINGS
.rodata1	SHT_PROGBITS	SHF_AL-LOC+SHF_MERGE+SH F_STRINGS
.shstrtab	SHT_STRTAB	0
.strtab	SHT_STRTAB	SHF_ALLOC
.symtab	SHT_SYMTAB	SHF_ALLOC
.tbss	SHT_NOBITS	SHF_AL-LOC+SHF_WRITE+SHF _TLS
.tdata	SHT_PROGBITS	SHF_AL-LOC+SHF_WRITE+SHF _TLS
.text	SHT_PROGBITS	SHF_ALLOC+SHF_EX-ECINSTR

.bss

This section holds data that contributes to the program's memory image. The program may treat this data as uninitialized. However, the system shall initialize this data with zeroes when the program begins to run. The section occupies no file space, as indicated by the section type, SHT_NOBITS.

.comment

This section holds version control information.

.data

This section holds initialized data that contribute to the program's memory image.

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

.data1

This section holds initialized data that contribute to the program's memory image.

.debug

This section holds information for symbolic debugging. The contents are unspecified. All section names with the prefix .debug hold information for symbolic debugging. The contents of these sections are unspecified.

.dynamic

This section holds dynamic linking information. The section's attributes will include the SHF_ALLOC bit. Whether the SHF_WRITE bit is set is processor specific. See Chapter 5 of System V ABI Update for more information.

.dynstr

This section holds strings needed for dynamic linking, most commonly the strings that represent the names associated with symbol table entries. See Chapter 5 of System V ABI Update for more information.

.dynsym

This section holds the dynamic linking symbol table, as described in `Symbol Table' of System V ABI Update.

.fini

This section holds executable instructions that contribute to the process termination code. That is, when a program exits normally, the system arranges to execute the code in this section.

.fini_array

This section holds an array of function pointers that contributes to a single termination array for the executable or shared object containing the section.

.hash

This section holds a symbol hash table. See `Hash Table' in Chapter 5 of System V ABI Update for more information.

.init

This section holds executable instructions that contribute to the process initialization code. When a program starts to run, the system arranges to execute the code in this section before calling the main program entry point (called main for C programs).

.init_array

This section holds an array of function pointers that contributes to a single initialization array for the executable or shared object containing the section.

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

.interp

This section holds the path name of a program interpreter. If the file has a loadable segment that includes relocation, the sections' attributes will include the SHF_ALLOC bit; otherwise, that bit will be off. See Chapter 5 of System V ABI Update for more information.

.line

This section holds line number information for symbolic debugging, which describes the correspondence between the source program and the machine code. The contents are unspecified.

.note

This section holds information in the format that `Note Section' in Chapter 5 of System V ABI Update describes.

.preinit_array

This section holds an array of function pointers that contributes to a single pre-initialization array for the executable or shared object containing the section.

.rodata

This section holds read-only data that typically contribute to a non-writable segment in the process image. See `Program Header' in Chapter 5 of System V ABI Update for more information.

.rodata1

This section holds read-only data that typically contribute to a non-writable segment in the process image. See `Program Header' in Chapter 5 of System V ABI Update for more information.

.shstrtab

This section holds section names.

.strtab

This section holds strings, most commonly the strings that represent the names associated with symbol table entries. If the file has a loadable segment that includes the symbol string table, the section's attributes will include the SHF_ALLOC bit; otherwise, that bit will be off.

.symtab

This section holds a symbol table, as `Symbol Table' in Chapter 4 of System V ABI Update describes. If the file has a loadable segment that includes the symbol table, the section's attributes will include the SHF_ALLOC bit; otherwise, that bit will be off.

.tbss

This section holds uninitialized thread-local data that contribute to the program's memory image. By definition, the system initializes the data with zeros when the data is instantiated for each new execution flow. The section

AVA Documents Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

occupies no file space, as indicated by the section type, SHT_NOBITS. Implementations need not support thread-local storage.

.tdata

This section holds initialized thread-local data that contributes to the program's memory image. A copy of its contents is instantiated by the system for each new execution flow. Implementations need not support thread-local storage.

.text

This section holds the `text', or executable instructions, of a program.

10.3.1.2 Additional Special Sections

Object files in an LSB conforming application may also contain one or more of the additional special sections described below.

Table 10-4 Additional Special Sections

Name	Type	Attributes
.ctors	SHT_PROGBITS	SHF_AL-LOC+SHF_WRITE
.data.rel.ro	SHT_PROGBITS	SHF_AL-LOC+SHF_WRITE
.dtors	SHT_PROGBITS	SHF_AL-LOC+SHF_WRITE
.eh_frame	SHT_PROGBITS	SHF_ALLOC
.eh_frame_hdr	SHT_PROGBITS	SHF_ALLOC
.gcc_except_table	SHT_PROGBITS	SHF_ALLOC
.gnu.version	SHT_GNU_versym	SHF_ALLOC
.gnu.version_d	SHT_GNU_verdef	SHF_ALLOC
.gnu.version_r	SHT_GNU_verneed	SHF_ALLOC
.got.plt	SHT_PROGBITS	SHF_AL-LOC+SHF_WRITE
.jcr	SHT_PROGBITS	SHF_AL-LOC+SHF_WRITE
.note.ABI-tag	SHT_NOTE	SHF_ALLOC
.stab	SHT_PROGBITS	0
.stabstr	SHT_STRTAB	0

.ctors

This section contains a list of global constructor function pointers.

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

.data.rel.ro

This section holds initialized data that contribute to the program's memory image. This section may be made read-only after relocations have been applied.

.dtors

This section contains a list of global destructor function pointers.

.eh_frame

This section contains information necessary for frame unwinding during exception handling. See Section 10.6.1.

.eh_frame_hdr

This section contains a pointer to the .eh_frame section which is accessible to the runtime support code of a C++ application. This section may also contain a binary search table which may be used by the runtime support code to more efficiently access records in the .eh_frame section. See Section 10.6.2.

.gcc_except_table

This section holds Language Specific Data.

.gnu.version

This section contains the Symbol Version Table. See Section 10.7.2.

.gnu.version_d

This section contains the Version Definitions. See Section 10.7.3.

.gnu.version_r

This section contains the Version Requirements. See Section 10.7.4.

.got.plt

This section holds the read-only portion of the GLobal Offset Table. This section may be made read-only after relocations have been applied.

.jcr

This section contains information necessary for registering compiled Java classes. The contents are compiler-specific and used by compiler initialization functions.

.note.ABI-tag

Specify ABI details. See Section 10.8.

.stab

This section contains debugging information. The contents are not specified as part of the LSB.

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

.stabstr

This section contains strings associated with the debugging infomation contained in the .stab section.

10.4 Symbol Mapping

10.4.1 Introduction

Symbols in a source program are translated by the compilation system into symbols that exist in the object file.

10.4.1.1 C Language

External C symbols shall be unchanged in an object file's symbol table.

10.5 DWARF Extensions

The LSB does not specify debugging information, however, some additional sections contain information which is encoded using the the encoding as specified by DWARF Debugging Information Format, Version 4 with extensions defined here.

10.5.1 DWARF Exception Header Encoding

The DWARF Exception Header Encoding is used to describe the type of data used in the `.eh_frame` and `.eh_frame_hdr` section. The upper 4 bits indicate how the value is to be applied. The lower 4 bits indicate the format of the data.

Table 10-5 DWARF Exception Header value format

Name	Value	Meaning
DW_EH_PE_absptr	0x00	The Value is a literal pointer whose size is determined by the architecture.
DW_EH_PE_uleb128	0x01	Unsigned value is encoded using the Little Endian Base 128 (LEB128) as defined by DWARF Debugging Information Format, Version 4.
DW_EH_PE_udata2	0x02	A 2 bytes unsigned value.
DW_EH_PE_udata4	0x03	A 4 bytes unsigned value.
DW_EH_PE_udata8	0x04	An 8 bytes unsigned value.
DW_EH_PE_sleb128	0x09	Signed value is encoded using the Little Endian

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Name	Value	Meaning
		Base 128 (LEB128) as defined by DWARF Debugging Information Format, Version 4.
DW_EH_PE_sdata2	0x0A	A 2 bytes signed value.
DW_EH_PE_sdata4	0x0B	A 4 bytes signed value.
DW_EH_PE_sdata8	0x0C	An 8 bytes signed value.

Table 10-6 DWARF Exception Header application

Name	Value	Meaning
DW_EH_PE_pcrel	0x10	Value is relative to the current program counter.
DW_EH_PE_textrel	0x20	Value is relative to the beginning of the .text section.
DW_EH_PE_datarel	0x30	Value is relative to the beginning of the .got or .eh_frame_hdr section.
DW_EH_PE_funcrel	0x40	Value is relative to the beginning of the function.
DW_EH_PE_aligned	0x50	Value is aligned to an address unit sized boundary.

One special encoding, 0xff (DW_EH_PE_omit), shall be used to indicate that no value ispresent.

10.5.2 DWARF CFI Extensions

In addition to the Call Frame Instructions defined in section 6.4.2 of DWARF Debugging Information Format, Version 4, the following additional Call Frame Instructions may also be used.

Table 10-7 Additional DWARF Call Frame Instructions

Name	Value	Meaning
DW_CFA_GNU_args_size	0x2e	The DW_CFA_GNU_args_size instruction takes an unsigned LEB128 operand representing

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

Name	Value	Meaning
		an argument size. This instruction specifies the total of the size of the arguments which have been pushed onto the stack.
DW_CFA_GNU_negative_offset_extended	0x2f	The DW_CFA_def_cfa_sf instruction takes two operands: an unsigned LEB128 value representing a register number and an unsigned LEB128 which represents the magnitude of the offset. This instruction is identical to DW_CFA_offset_extended_sf except that the operand is subtracted to produce the offset. This instructions is obsoleted by DW_CFA_offset_extended_sf.

10.6 Exception Frames

When using languages that support exceptions, such as C++, additional information must be provided to the runtime environment that describes the call frames that must be unwound during the processing of an exception. This information is contained in the special sections `.eh_frame` and `.eh_framehdr`.

> **Note:** The format of the `.eh_frame` section is similar in format and purpose to the `.debug_frame` section which is specified in DWARF Debugging Information Format, Version 4. Readers are advised that there are some subtle difference, and care should be taken when comparing the two sections.

10.6.1 The `.eh_frame` section

The `.eh_frame` section shall contain 1 or more Call Frame Information (CFI) records. The number of records present shall be determined by size of the section as contained in the section header. Each CFI record contains a Common Information Entry (CIE) record followed by 1 or more Frame Description Entry (FDE) records. Both CIEs and FDEs shall be aligned to an addressing unit sized boundary.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Table 10-8 Call Frame Information Format

Common Information Entry Record
Frame Description Entry Record(s)

10.6.1.1 The Common Information Entry Format

Table 10-9 Common Information Entry Format

Length	Required
Extended Length	Optional
CIE ID	Required
Version	Required
Augmentation String	Required
Code Alignment Factor	Required
Data Alignment Factor	Required
Return Address Register	Required
Augmentation Data Length	Optional
Augmentation Data	Optional
Initial Instructions	Required
Padding	

Length

A 4 byte unsigned value indicating the length in bytes of the CIE structure, not including the *Length* field itself. If *Length* contains the value 0xffffffff, then the length is contained in the *Extended Length* field. If *Length* contains the value 0, then this CIE shall be considered a terminator and processing shall end.

Extended Length

A 8 byte unsigned value indicating the length in bytes of the CIE structure, not including the *Length* and *Extended Length* fields themselves. This field is not present unless the *Length* field contains the value 0xffffffff.

CIE ID

A 4 byte unsigned value that is used to distinguish CIE records from FDE records. This value shall always be 0, which indicates this record is a CIE.

Version

A 1 byte value that identifies the version number of the frame information structure. This value shall be 1.

Augmentation String

This value is a NUL terminated string that identifies the augmentation to the CIE or to the FDEs associated with this CIE. A zero length string indicates

AVA Documents Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

that no augmentation data is present. The augmentation string is case sensitive and shall be interpreted as described below.

Code Alignment Factor

An unsigned LEB128 encoded value that is factored out of all advance location instructions that are associated with this CIE or its FDEs. This value shall be multiplied by the delta argument of an adavance location instruction to obtain the new location value.

Data Alignment Factor

A signed LEB128 encoded value that is factored out of all offset instructions that are associated with this CIE or its FDEs. This value shall be multiplied by the register offset argument of an offset instruction to obtain the new offset value.

Augmentation Length

An unsigned LEB128 encoded value indicating the length in bytes of the Augmentation Data. This field is only present if the Augmentation String contains the character 'z'.

Augmentation Data

A block of data whose contents are defined by the contents of the Augmentation String as described below. This field is only present if the Augmentation String contains the character 'z'. The size of this data is given by the Augentation Length.

Initial Instructions

Initial set of Call Frame Instructions. The number of instructions is determined by the remaining space in the CIE record.

Padding

Extra bytes to align the CIE structure to an addressing unit size boundary.

10.6.1.1.1 Augmentation String Format

The Agumentation String indicates the presence of some optional fields, and how those fields should be intepreted. This string is case sensitive. Each character in the augmentation string in the CIE can be interpreted as below:

'z'

A 'z' may be present as the first character of the string. If present, the Augmentation Data field shall be present. The contents of the Augmentation Data shall be intepreted according to other characters in the Augmentation String.

'L'

A 'L' may be present at any position after the first character of the string. This character may only be present if 'z' is the first character of the string. If present, it indicates the presence of one argument in the Augmentation Data of the CIE, and a corresponding argument in the Augmentation Data of the FDE. The argument in the Augmentation Data of the CIE is 1-byte and

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

represents the pointer encoding used for the argument in the Augmentation Data of the FDE, which is the address of a language-specific data area (LSDA). The size of the LSDA pointer is specified by the pointer encoding used.

'P'

A 'P' may be present at any position after the first character of the string. This character may only be present if 'z' is the first character of the string. If present, it indicates the presence of two arguments in the Augmentation Data of the CIE. The first argument is 1-byte and represents the pointer encoding used for the second argument, which is the address of a *personality routine* handler. The personality routine is used to handle language and vendor-specific tasks. The system unwind library interface accesses the language-specific exception handling semantics via the pointer to the personality routine. The personality routine does not have an ABI-specific name. The size of the personality routine pointer is specified by the pointer encoding used.

'R'

A 'R' may be present at any position after the first character of the string. This character may only be present if 'z' is the first character of the string. If present, The Augmentation Data shall include a 1 byte argument that represents the pointer encoding for the address pointers used in the FDE.

10.6.1.2 The Frame Description Entry Format

Table 10-10 Frame Description Entry Format

Length	Required
Extended Length	Optional
FDE Pointer	Required
PC Begin	Required
PC Range	Required
Augmentation Data Length	Optional
Augmentation Data	Optional
Call Frame Instructions	Required
Padding	

`Length`

A 4 byte unsigned value indicating the length in bytes of the FDE structure, not including the `Length` field itself. If `Length` contains the value 0xffffffff, then the length is contained the `Extended Length` field. If `Length` contains the value 0, then this FDE shall be considered a terminator and processing shall end.

© 2021 ISO/IEC – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Extended Length

A 8 byte unsigned value indicating the length in bytes of the FDE structure, not including the *Length* or *Extended Length* field themselves. This field is not present unless the *Length* field contains the value 0xffffffff.

CIE Pointer

A 4 byte unsigned value that when subtracted from the offset of the CIE Pointer in the current FDE yields the offset of the start of the associated CIE. This value shall never be 0.

PC Begin

An encoded value that indicates the address of the initial location associated with this FDE. The encoding format is specified in the Augmentation Data.

PC Range

An absolute value that indicates the number of bytes of instructions associated with this FDE.

Augmentation Length

An unsigned LEB128 encoded value indicating the length in bytes of the Augmentation Data. This field is only present if the Augmentation String in the associated CIE contains the character 'z'.

Augmentation Data

A block of data whose contents are defined by the contents of the Augmentation String in the associated CIE as described above. This field is only present if the Augmentation String in the associated CIE contains the character 'z'. The size of this data is given by the Augentation Length.

Call Frame Instructions

A set of Call Frame Instructions.

Padding

Extra bytes to align the FDE structure to an addressing unit size boundary.

10.6.2 The `.eh_frame_hdr` section

The `.eh_frame_hdr` section contains additional information about the `.eh_frame` section. A pointer to the start of the `.eh_frame` data, and optionally, a binary search table of pointers to the `.eh_frame` records are found in this section.

Data in this section is encoded according to Section 10.5.1.

Table 10-11 `.eh_frame_hdr` Section Format

Encoding	Field
unsigned byte	version
unsigned byte	eh_frame_ptr_enc
unsigned byte	fde_count_enc

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

Encoding	Field
unsigned byte	table_enc
encoded	eh_frame_ptr
encoded	fde_count
	binary search table

version

Version of the `.eh_frame_hdr` format. This value shall be 1.

eh_frame_ptr_enc

The encoding format of the eh_frame_ptr field.

fde_count_enc

The encoding format of the fde_count field. A value of DW_EH_PE_omit indicates the binary search table is not present.

table_enc

The encoding format of the entries in the binary search table. A value of DW_EH_PE_omit indicates the binary search table is not present.

eh_frame_ptr

The encoded value of the pointer to the start of the `.eh_frame` section.

fde_count

The encoded value of the count of entries in the binary search table.

binary search table

A binary search table containing fde_count entries. Each entry of the table consist of two encoded values, the initial location, and the address. The entries are sorted in an increasing order by the initial location value.

10.7 Symbol Versioning

10.7.1 Introduction

This chapter describes the Symbol Versioning mechanism. All ELF objects may provide or depend on versioned symbols. Symbol Versioning is implemented by 3 section types: `SHT_GNU_versym`, `SHT_GNU_verdef`, and `SHT_GNU_verneed`.

The prefix `Elfxx` in the following descriptions and code fragments stands for either "`Elf32`" or "`Elf64`", depending on the architecture.

Versions are described by strings. The structures that are used for symbol versions also contain a member that holds the ELF hashing values of the strings. This allows for more efficient processing.

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021

10.7.2 Symbol Version Table

The special section `.gnu.version` which has a section type of `SHT_GNU_versym` shall contain the Symbol Version Table. This section shall have the same number of entries as the Dynamic Symbol Table in the `.dynsym` section.

The `.gnu.version` section shall contain an array of elements of type `Elfxx_Half`. Each entry specifies the version defined for or required by the corresponding symbol in the Dynamic Symbol Table.

The values in the Symbol Version Table are specific to the object in which they are located. These values are identifiers that are provided by the the *`vna_other`* member of the `Elfxx_Vernaux` structure or the *`vd_ndx`* member of the `Elfxx_Verdef` structure.

The values 0 and 1 are reserved.

0

The symbol is local, not available outside the object.

1

The symbol is defined in this object and is globally available.

All other values are used to identify version strings located in one of the other Symbol Version sections. The value itself is not the version associated with the symbol. The string identified by the value defines the version of the symbol.

10.7.3 Version Definitions

The special section `.gnu.version_d` which has a section type of `SHT_GNU_verdef` shall contain symbol version definitions. The number of entries in this section shall be contained in the `DT_VERDEFNUM` entry of the Dynamic Section `.dynamic`. The `sh_link` member of the section header (see figure 4-8 in the System V ABI) shall point to the section that contains the strings referenced by this section.

The section shall contain an array of `Elfxx_Verdef` structures, as described in Figure 10-1, optionally followed by an array of `Elfxx_Verdaux` structures, as defined in Figure 10-2.

```
typedef struct {
       Elfxx_Half    vd_version;
       Elfxx_Half    vd_flags;
       Elfxx_Half    vd_ndx;
       Elfxx_Half    vd_cnt;
       Elfxx_Word    vd_hash;
       Elfxx_Word    vd_aux;
       Elfxx_Word    vd_next;
} Elfxx_Verdef;
```

Figure 10-1 Version Definition Entries

`vd_version`

Version revision. This field shall be set to `1`.

`vd_flags`

Version information flag bitmask.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

vd_ndx

Version index numeric value referencing the SHT_GNU_versym section.

vd_cnt

Number of associated verdaux array entries.

vd_hash

Version name hash value (ELF hash function).

vd_aux

Offset in bytes to a corresponding entry in an array of `Elfxx_Verdaux` structures as defined in Figure 10-2

vd_next

Offset to the next verdef entry, in bytes.

```
typedef struct {
       Elfxx_Word    vda_name;
       Elfxx_Word    vda_next;
} Elfxx_Verdaux;
```

Figure 10-2 Version Definition Auxiliary Entries

vda_name

Offset to the version or dependency name string in the section header, in bytes.

vda_next

Offset to the next verdaux entry, in bytes.

10.7.4 Version Requirements

The special section `.gnu.version_r` which has a section type of `SHT_GNU_verneed` shall contain required symbol version definitions. The number of entries in this section shall be contained in the `DT_VERNEEDNUM` entry of the Dynamic Section `.dynamic`. The *sh_link* member of the section header (see figure 4-8 in System V ABI) shall point to the section that contains the strings referenced by this section.

The section shall contain an array of `Elfxx_Verneed` structures, as described in Figure 10-3, optionally followed by an array of `Elfxx_Vernaux` structures, as defined in Figure 10-4.

```
typedef struct {
       Elfxx_Half    vn_version;
       Elfxx_Half    vn_cnt;
       Elfxx_Word    vn_file;
       Elfxx_Word    vn_aux;
       Elfxx_Word    vn_next;
} Elfxx_Verneed;
```

Figure 10-3 Version Needed Entries

vn_version

Version of structure. This value is currently set to 1, and will be reset if the versioning implementation is incompatibly altered.

© 2021 ISO/IEC – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

vn_cnt

Number of associated verneed array entries.

vn_file

Offset to the file name string in the section header, in bytes.

vn_aux

Offset to a corresponding entry in the vernaux array, in bytes.

vn_next

Offset to the next verneed entry, in bytes.

```
typedef struct {
       Elfxx_Word    vna_hash;
       Elfxx_Half    vna_flags;
       Elfxx_Half    vna_other;
       Elfxx_Word    vna_name;
       Elfxx_Word    vna_next;
} Elfxx_Vernaux;
```

Figure 10-4 Version Needed Auxiliary Entries

vna_hash

Dependency name hash value (ELF hash function).

vna_flags

Dependency information flag bitmask.

vna_other

Object file version identifier used in the .gnu.version symbol version array. Bit number 15 controls whether or not the object is hidden; if this bit is set, the object cannot be used and the static linker will ignore the symbol's presence in the object.

vna_name

Offset to the dependency name string in the section header, in bytes.

vna_next

Offset to the next vernaux entry, in bytes.

10.7.5 Startup Sequence

When loading a sharable object the system shall analyze version definition data from the loaded object to assure that it meets the version requirements of the calling object. This step is referred to as definition testing. The dynamic loader shall retrieve the entries in the caller's `Elfxx_Verneed` array and attempt to find matching definition information in the loaded `Elfxx_Verdef` table.

Each object and dependency shall be tested in turn. If a symbol definition is missing and the `vna_flags` bit for `VER_FLG_WEAK` is not set, the loader shall return an error and exit. If the `vna_flags` bit for `VER_FLG_WEAK` is set in the `Elfxx_Vernaux` entry, and the loader shall issue a warning and continue operation.

© 2021 ISO/IEC – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23360-1-2:2021

When the versions referenced by undefined symbols in the loaded object are found, version availability is certified. The test completes without error and the object shall be made available.

10.7.6 Symbol Resolution

When symbol versioning is used in an object, relocations extend definition testing beyond the simple match of symbol name strings: the version of the reference shall also equal the name of the definition.

The same index that is used in the symbol table can be referenced in the `SHT_GNU_versym` section, and the value of this index is then used to acquire name data. The corresponding requirement string is retrieved from the `Elfxx_Verneed` array, and likewise, the corresponding definition string from the `Elfxx_Verdef` table.

If the high order bit (bit number 15) of the version symbolis set, the object cannot be used and the static linker shall ignore the symbol's presence in the object.

When an object with a reference and an object with the definition are being linked, the following rules shall govern the result:

- The object with the reference and the object with the definitions both use versioning. All described matching is processed in this case. A fatal error shall be triggered when no matching definition can be found in the object whose name is the one referenced by the *`vn_name`* element in the `Elfxx_Verneed` entry.
- The object with the reference does not use versioning, while the object with the definitions does. In this instance, only the definitions with index numbers 1 and 2 will be used in the reference match, the same identified by the static linker as the base definition. In cases where the static linker was not used, such as in calls to `dlopen()`, a version that does not have the base definition index shall be acceptable if it is the only version for which the symbol is defined.
- The object with the reference uses versioning, but the object with the definitions specifies none. A matching symbol shall be accepted in this case. A fatal error shall be triggered if a corruption in the required symbols list obscures an outdated object file and causes a match on the object filename in the `Elfxx_Verneed` entry.
- Neither the object with the reference nor the object with the definitions use versioning. The behavior in this instance shall default to pre-existing symbol rules.

10.8 ABI note tag

Every executable shall contain a section named `.note.ABI-tag` of type `SHT_NOTE`. This section is structured as a note section as documented in the ELF spec. The section shall contain at least the following entry. The `name` field (`namesz`/`name`) contains the string "`GNU`". The `type` field shall be 1. The `descsz` field shall be at least 16, and the first 16 bytes of the `desc` field shall be as follows.

The first 32-bit word of the `desc` field shall be 0 (this signifies a Linux executable). The second, third, and fourth 32-bit words of the `desc` field contain the earliest compatible kernel version. For example, if the 3 words are 2, 2, and 5, this signifies a 2.2.5 kernel.

Documents
Document Sharing Hub

© 2021 ISO/IEC – All rights reserved

Full document details: BS ISO/IEC 23360-1-2:2021