

---

# **C++ ABI for the Arm 64-bit Architecture (AArch64)**

*Release 2019Q4*

**Arm Ltd**

**Jan 31, 2020**

# CONTENTS

<b>1</b>	<b>Preamble</b>	<b>2</b>
1.1	Abstract . . . . .	2
1.2	Keywords . . . . .	2
1.3	How to find the latest release of this specification or report a defect in it . . . . .	2
1.4	Licence . . . . .	2
1.5	Non-Confidential Proprietary Notice . . . . .	2
1.6	Contents . . . . .	3
<b>2</b>	<b>About this document</b>	<b>5</b>
2.1	Change Control . . . . .	5
2.2	Change History . . . . .	5
2.3	References . . . . .	5
2.4	Terms and Abbreviations . . . . .	6
2.5	Your licence to use this specification . . . . .	7
<b>3</b>	<b>Overview</b>	<b>9</b>
3.1	The Generic C++ ABI . . . . .	9
3.2	The Exception Handling ABI for the Arm architecture . . . . .	9
<b>4</b>	<b>The C++ ABI Supplement</b>	<b>11</b>
4.1	Summary of differences from and additions to the generic C++ ABI . . . . .	11
4.2	Differences in detail . . . . .	12
<b>5</b>	<b>EH ABI Level III: Implementation ABI for GNU Linux</b>	<b>15</b>
5.1	Introduction . . . . .	15
5.2	Data Structures . . . . .	15
5.3	Standard Runtime Initialization . . . . .	15
5.4	Throwing an Exception . . . . .	15
5.5	Catching an Exception . . . . .	16

Document number: IHI 0059D, current through AArch64 ABI release 2019Q4

Date of Issue: 30<sup>th</sup> January 2020

---

## PREAMBLE

### 1.1 Abstract

This document describes the C++ Application Binary Interface for the Arm 64-bit architecture.

### 1.2 Keywords

C++, Application Binary Interface, ABI, AArch64, C++ ABI, generic C++ ABI

### 1.3 How to find the latest release of this specification or report a defect in it

Please check the Arm Developer site (<https://developer.arm.com/products/software-development-tools/specifications>) for a later release if your copy is more than one year old.

Please report defects in this specification to *arm dot eabi at arm dot com*.

### 1.4 Licence

THE TERMS OF YOUR ROYALTY FREE LIMITED LICENCE TO USE THIS ABI SPECIFICATION ARE GIVEN IN *Your licence to use this specification* (page 7) (Arm contract reference LEC-ELA-00081 V2.0). PLEASE READ THEM CAREFULLY.

BY DOWNLOADING OR OTHERWISE USING THIS SPECIFICATION, YOU AGREE TO BE BOUND BY ALL OF ITS TERMS. IF YOU DO NOT AGREE TO THIS, DO NOT DOWNLOAD OR USE THIS SPECIFICATION. THIS ABI SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES (SEE *Your licence to use this specification* (page 7) FOR DETAILS).

### 1.5 Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2010, 2020 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England. 110 Fulbourn Road, Cambridge, England CB1 9NJ. LES-PRE-20349

## 1.6 Contents

### Contents

- C++ Application Binary Interface Standard for the Arm® 64-bit Architecture (page ??)
  - *Preamble* (page 2)
    - \* *Abstract* (page 2)
    - \* *Keywords* (page 2)
    - \* *How to find the latest release of this specification or report a defect in it* (page 2)
    - \* *Licence* (page 2)
    - \* *Non-Confidential Proprietary Notice* (page 2)
    - \* *Contents* (page 3)
  - *About this document* (page 5)
    - \* *Change Control* (page 5)

- \* *Change History* (page 5)
- \* *References* (page 5)
- \* *Terms and Abbreviations* (page 6)
- \* *Your licence to use this specification* (page 7)
- *Overview* (page 9)
  - \* *The Generic C++ ABI* (page 9)
  - \* *The Exception Handling ABI for the Arm architecture* (page 9)
- *The C++ ABI Supplement* (page 11)
  - \* *Summary of differences from and additions to the generic C++ ABI* (page 11)
  - \* *Differences in detail* (page 12)
- *EH ABI Level III: Implementation ABI for GNU Linux* (page 15)
  - \* *Introduction* (page 15)
  - \* *Data Structures* (page 15)
  - \* *Standard Runtime Initialization* (page 15)
  - \* *Throwing an Exception* (page 15)
  - \* *Catching an Exception* (page 16)

## ABOUT THIS DOCUMENT

### 2.1 Change Control

#### 2.1.1 Current Status and Anticipated Changes

The following support level definitions are used by the Arm ABI specifications:

**Release** Arm considers this specification to have enough implementations, which have received sufficient testing, to verify that it is correct. The details of these criteria are dependent on the scale and complexity of the change over previous versions: small, simple changes might only require one implementation, but more complex changes require multiple independent implementations, which have been rigorously tested for cross-compatibility. Arm anticipates that future changes to this specification will be limited to typographical corrections, clarifications and compatible extensions.

**Beta** Arm considers this specification to be complete, but existing implementations do not meet the requirements for confidence in its release quality. Arm may need to make incompatible changes if issues emerge from its implementation.

**Alpha** The content of this specification is a draft, and Arm considers the likelihood of future incompatible changes to be significant.

All content in this document is at the **Release** quality level.

### 2.2 Change History

Issue	Date	By	Change
00bet3	15th December 2010	MGD	Beta release.
1.0	22nd May 2013	RE	First public release.
2018Q4	31st December 2018	OS	Typographical changes.
2019Q4	30st January 2020	TS	Add name mangling rules for half-precision Brain floating point format: <i>Summary of differences from and additions to the generic C++ ABI</i> (page 11).

### 2.3 References

This document refers to, or is referred to by, the following documents.

Ref	URL or other reference	Title
AAPCS64	IHI 0055	Procedure Call Standard for the Arm 64-bit Architecture
AAELF64	IHI 0056	ELF for the Arm 64-bit Architecture

Continued on next page

Table 2.2 – continued from previous page

Ref	URL or other reference	Title
<a href="#">CPPABI64</a> (page 1)	This document	C++ ABI for the Arm 64-bit Architecture
GC++ABI	<a href="http://itanium-cxx-abi.github.io/cxx-abi/abi.html">http://itanium-cxx-abi.github.io/cxx-abi/abi.html</a>	Generic C++ ABI
Generic ELF	<a href="http://www.sco.com/developers/gabi/">http://www.sco.com/developers/gabi/</a>	Generic ELF, 17th December 2003 Draft
ISO C++	ISO/IEC 14882:2003 (14882:1988 with Technical Corrigendum)	International Standard ISO/IEC 14882:2003 – Programming languages C++
LSB <sup>1</sup>	<a href="https://refspecs.linuxfoundation.org/LSB_4.0.0/LSB-Core-generic/LSB-Core-generic.html">https://refspecs.linuxfoundation.org/LSB_4.0.0/LSB-Core-generic/LSB-Core-generic.html</a>	Linux Standards Base Core Specification 4.0
IA64EHABI <sup>2</sup>	<a href="https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html">https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html</a>	Itanium C++ ABI: Exception Handling

## 2.4 Terms and Abbreviations

The ABI for the Arm 64-bit Architecture uses the following terms and abbreviations.

**A32** The instruction set named Arm in the Armv7 architecture; A32 uses 32-bit fixed-length instructions.

**A64** The instruction set available when in AArch64 state.

**AAPCS64** Procedure Call Standard for the Arm 64-bit Architecture (AArch64)

**AArch32** The 32-bit general-purpose register width state of the Armv8 architecture, broadly compatible with the Armv7-A architecture.

**AArch64** The 64-bit general-purpose register width state of the Armv8 architecture.

**ABI** Application Binary Interface:

1. The specifications to which an executable must conform in order to execute in a specific execution environment. For example, the *Linux ABI for the Arm Architecture*.
2. A particular aspect of the specifications to which independently produced relocatable files must conform in order to be statically linkable and executable. For example, the *C++ ABI for the Arm Architecture* (page 1), ELF for the Arm Architecture, ...

**Arm-based** ... based on the Arm architecture ...

**Floating point** Depending on context floating point means or qualifies: (a) floating-point arithmetic conforming to IEEE 754 2008; (b) the Armv8 floating point instruction set; (c) the register set shared by (b) and the Armv8 SIMD instruction set.

**Q-o-I** Quality of Implementation – a quality, behavior, functionality, or mechanism not required by this standard, but which might be provided by systems conforming to it. Q-o-I is often used to describe the tool-chain-specific means by which a standard requirement is met.

**SIMD** Single Instruction Multiple Data – A term denoting or qualifying: (a) processing several data items in parallel under the control of one instruction; (b) the Arm v8 SIMD instruction set; (c) the register set shared by (b) and the Armv8 floating point instruction set.

**SIMD and floating point** The Arm architecture's SIMD and Floating Point architecture comprising the floating point instruction set, the SIMD instruction set and the register set shared by them.

<sup>1</sup> [https://refspecs.linuxfoundation.org/LSB\\_4.0.0/LSB-Core-generic/LSB-Core-generic.html](https://refspecs.linuxfoundation.org/LSB_4.0.0/LSB-Core-generic/LSB-Core-generic.html)

<sup>2</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>



**T32** The instruction set named Thumb in the Armv7 architecture; T32 uses 16-bit and 32-bit instructions.

More specific terminology is defined when it is first used.

## 2.5 Your licence to use this specification

**IMPORTANT: THIS IS A LEGAL AGREEMENT (“LICENCE”) BETWEEN YOU (AN INDIVIDUAL OR SINGLE ENTITY WHO IS RECEIVING THIS DOCUMENT DIRECTLY FROM ARM LIMITED) (“LICENSEE”) AND ARM LIMITED (“ARM”) FOR THE SPECIFICATION DEFINED IMMEDIATELY BELOW. BY DOWNLOADING OR OTHERWISE USING IT, YOU AGREE TO BE BOUND BY ALL OF THE TERMS OF THIS LICENCE. IF YOU DO NOT AGREE TO THIS, DO NOT DOWNLOAD OR USE THIS SPECIFICATION.**

“Specification” means, and is limited to, the version of the specification for the Applications Binary Interface for the Arm Architecture comprised in this document. Notwithstanding the foregoing, “Specification” shall not include (i) the implementation of other published specifications referenced in this Specification; (ii) any enabling technologies that may be necessary to make or use any product or portion thereof that complies with this Specification, but are not themselves expressly set forth in this Specification (e.g. compiler front ends, code generators, back ends, libraries or other compiler, assembler or linker technologies; validation or debug software or hardware; applications, operating system or driver software; RISC architecture; processor microarchitecture); (iii) maskworks and physical layouts of integrated circuit designs; or (iv) RTL or other high level representations of integrated circuit designs.

Use, copying or disclosure by the US Government is subject to the restrictions set out in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software – Restricted Rights at 48 C.F.R. 52.227-19, as applicable.

This Specification is owned by Arm or its licensors and is protected by copyright laws and international copyright treaties as well as other intellectual property laws and treaties. The Specification is licensed not sold.

1. Subject to the provisions of Clauses 2 and 3, Arm hereby grants to LICENSEE, under any intellectual property that is (i) owned or freely licensable by Arm without payment to unaffiliated third parties and (ii) either embodied in the Specification or Necessary to copy or implement an applications binary interface compliant with this Specification, a perpetual, non-exclusive, non-transferable, fully paid, worldwide limited licence (without the right to sublicense) to use and copy this Specification solely for the purpose of developing, having developed, manufacturing, having manufactured, offering to sell, selling, supplying or otherwise distributing products which comply with the Specification.
2. THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF SATISFACTORY QUALITY, MERCHANTABILITY, NONINFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE. THE SPECIFICATION MAY INCLUDE ERRORS. Arm RESERVES THE RIGHT TO INCORPORATE MODIFICATIONS TO THE SPECIFICATION IN LATER REVISIONS OF IT, AND TO MAKE IMPROVEMENTS OR CHANGES IN THE SPECIFICATION OR THE PRODUCTS OR TECHNOLOGIES DESCRIBED THEREIN AT ANY TIME.
3. This Licence shall immediately terminate and shall be unavailable to LICENSEE if LICENSEE or any party affiliated to LICENSEE asserts any patents against Arm, Arm affiliates, third parties who have a valid licence from Arm for the Specification, or any customers or distributors of any of them based upon a claim that a LICENSEE (or LICENSEE affiliate) patent is Necessary to implement the Specification. In this Licence; (i) “affiliate” means any entity controlling, controlled by or under common control with a party (in fact or in law, via voting securities, management control or otherwise) and “affiliated” shall be construed accordingly; (ii) “assert” means to allege infringement in legal or administrative proceedings, or proceedings before any other competent trade, arbitral or international authority; (iii) “Necessary” means with respect to any claims of any patent, those claims which, without the appropriate permission of the patent owner, will be infringed when implementing the Specification because no alternative, commercially reasonable, non-infringing way

of implementing the Specification is known; and (iv) English law and the jurisdiction of the English courts shall apply to all aspects of this Licence, its interpretation and enforcement. The total liability of Arm and any of its suppliers and licensors under or in relation to this Licence shall be limited to the greater of the amount actually paid by LICENSEE for the Specification or US\$10.00. The limitations, exclusions and disclaimers in this Licence shall apply to the maximum extent allowed by applicable law.

Arm Contract reference LEC-ELA-00081 V2.0 AB/LS (9 March 2005)

## OVERVIEW

The *C++ ABI for the Arm 64-bit architecture* (page 1) comprises the following sub-components.

- The generic C++ ABI, summarized in *The Generic C++ ABI* (page 9), is the referenced base standard for this component.
- The C++ ABI supplement in *The C++ ABI Supplement* (page 11) details Arm-specific additions to and deviations from the generic standard.
- The generic C++ exception handling ABI summarized in *The Exception Handling ABI for the Arm architecture* (page 9), describes the language-independent and C++-specific aspects of exception handling.
- The C++ exception handling ABI supplement for GNU/Linux in *EH ABI Level III: Implementation ABI for GNU Linux* (page 15) details Arm-specific additions to and deviations from the generic standard for GNU/Linux systems.

The generic C++ ABI is implicitly an SVr4-based standard, and takes an SVr4 position on symbol visibility and vague linkage. This document does not cover extensions for DLL-based environments.

### 3.1 The Generic C++ ABI

The generic C++ ABI [GC++ABI] (originally developed for SVr4 on Itanium) specifies:

- The layout of C++ non-POD class types in terms of the layout of POD types (specified for this ABI by the Procedure Call Standard for the Arm Architecture [AAPCS64]).
- How class types requiring copy construction are passed as parameters and results.
- The content of run-time type information (RTTI).
- Necessary APIs for object construction and destruction.
- How names with linkage are mangled (name mangling).

The generic C++ ABI refers to a separate Itanium-specific specification of exception handling. When the generic C++ ABI is used as a component of this ABI, corresponding reference must be made to the generic C++ exception handling ABI as summarised in *The Exception Handling ABI for the Arm architecture* (page 9), and the C++ exception handling ABI supplement in *EH ABI Level III: Implementation ABI for GNU Linux* (page 15).

### 3.2 The Exception Handling ABI for the Arm architecture

In common with [IA64EHABI<sup>3</sup>], this ABI specifies table-based unwinding that separates language-independent unwinding from language specific aspects. The [IA64EHABI<sup>4</sup>] specification describes three levels of ABI:

---

<sup>3</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>

<sup>4</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>

Level I. A Base API, interfaces common to all languages and implementations.

Level II. The C++ ABI, interfaces for interoperability of C++ implementations.

Level III. The Implementation ABI specific to a particular runtime implementation.

The AArch64 C++ EH ABI uses Level I and Level II of the Itanium exception handling ABI as specified.

*EH ABI Level III: Implementation ABI for GNU Linux* (page 15) describes the EH Level III ABI used on GNU/Linux systems.

## THE C++ ABI SUPPLEMENT

### 4.1 Summary of differences from and additions to the generic C++ ABI

This section summarizes the differences between the C++ ABI for the Arm 64-bit architecture and the generic C++ ABI. Section numbers in captions refer to the generic C++ ABI specification. Larger differences are detailed in subsections of *Differences in detail* (page 12).

#### GC++ABI §2.2 POD Data Types

The GC++ABI defines the way in which empty class types are laid out. For the purposes of parameter passing in [AAPCS64], a parameter whose type is an empty class shall be treated as if its type were an aggregate with a single member of type unsigned byte.

---

**Note:** Of course, the single member has undefined content.

---

#### GC++ABI §2.3 Member Pointers

The pointer to member function representation differs from that used by Itanium. See *Representation of pointer to member function* (page 12).

#### GC++ABI §2.8 Initialization guard variables

This ABI only specifies the bottom bit of the guard variable. See *Guard variables* (page 12).

#### GC++ABI §3.1.4 Return Values

When a return value has a non-trivial copy constructor or destructor, the address of the caller-allocated temporary is passed in the Indirect Result Location Register (r8) rather than as an implicit first parameter before the this parameter and user parameters.

#### GC++ABI §3.3.4 Controlling Object Construction Order

Global object construction is managed in a simplified way under this ABI. See *Controlling Object Construction Order* (page 12).

#### GC++ABI §3.4 Demangler

This ABI provides the demangler interface as specified in the generic C++ ABI. The ABI does not specify the format of the demangled string.

#### GC++ABI §5.1.5 Builtin Types

The `__bf16` is mangled as `u6__bf16`.

#### GC++ABI §5.2.2 Static Data

If a static datum and its guard variable are emitted in the same COMDAT group, the ELF binding [Generic ELF] for both symbols must be `STB_GLOBAL`, not `STB_WEAK` as specified in [GC++ABI64]. *ELF binding of static data guard variable symbols* (page 13) justifies this requirement.

#### GC++ABI §5.3 Unwind Table Location

The exception unwind table shall be located by use of program header entries of type PT\_AARCH64\_UNWIND. See [Unwind Table Location](#) (page 14).

**(No section in the generic C++ ABI) A library nothrow new function must not examine its 2nd argument**

Library versions of the following functions must not examine their second argument.

```
::operator new(std::size_t, const std::nothrow_t&)\n::operator new[](std::size_t, const std::nothrow_t&)
```

(The second argument conveys no useful information other than through its presence or absence, which is manifest in the mangling of the name of the function. This ABI therefore allows code generators to use a potentially invalid second argument – for example, whatever value happens to be in R1 – at a point of call).

**(No section in the generic C++ ABI, but would be §2.2) POD data types**

Pointers to extern “C++” functions and pointers to extern “C” functions are interchangeable if the function types are otherwise identical.

In order to be used by the library helper functions described below, implementations of constructor and destructor functions (complete, sub-object, deleting, and allocating) must have a type compatible with:

```
extern "C" void (*)(void* /* , other argument types if any */);
```

**(No section in the generic C++ ABI) Namespace and mangling for the va\_list type**

The type `__va_list` is in namespace `std`. The type name of `va_list` therefore mangles to `St9__va_list`.

## 4.2 Differences in detail

### 4.2.1 Representation of pointer to member function

The generic C++ ABI [GC++ABI] specifies that a pointer to member function is a pair of words `<ptr, adj>`. The least significant bit of `ptr` discriminates between (0) the address of a non-virtual member function and (1) the offset in the class’s virtual table of the address of a virtual function.

This encoding cannot work for the AArch64 instruction set where the architecture reserves all bits of code addresses.

This ABI specifies that `adj` contains twice the `this` adjustment, plus 1 if the member function is virtual. The least significant bit of `adj` then makes exactly the same discrimination as the least significant bit of `ptr` does for Itanium.

A pointer to member function is NULL when `ptr = 0` and the least significant bit of `adj` is zero.

### 4.2.2 Guard variables

The generic C++ ABI [GC++ABI] specifies the bottom byte of a static variable guard variable shall be 0 when the variable is not initialized, and 1 when it is. All other bytes are platform defined.

This ABI instead only specifies the value bit 0 of the static guard variable; all other bits are platform defined. Bit 0 shall be 0 when the variable is not initialized and 1 when it is.

### 4.2.3 Controlling Object Construction Order

The generic ABI specifies a `#pragma` and `.priority_init` section type to allow the user to specify object construction order. This scheme is not in wide use and so this ABI uses a different scheme which has several pre-existing implementations.

The compiler is responsible for sequencing the construction of top-level static objects defined in a translation unit in accordance with the requirements of the C++ standard. The run-time environment (helper-function library) sequences the initialization of one translation unit after another. The global constructor vector provides the interface between these agents as follows:

- Each translation unit provides a fragment of the constructor vector in an ELF section called `.init_array` of type `SHT_INIT_ARRAY` (`=0xE`) and section flags `SHF_ALLOC + SHF_WRITE`.
- Each element of the vector contains the address of a function of type extern “C” void (\* const)(void) that, when called, performs part or all of the global object construction for the translation unit. Producers must treat `.init_array` sections as if they were read-only.
- The appropriate entry for an element referring to, say, `__sti_file` that constructs the global static objects in `filecpp`, is 0 relocated by `R_AARCH64_ABS64` (`__sti_file`).
- Object construction order may be controlled by appending an unsigned integer in the range 0-65535 (formatted as if by `printf(“%05d”, priority)`) to the name of the section. The linker must lay these sections out in ascending lexicographical order.
- Sections without a priority number appended are assumed to have a lower priority than those sections with a priority number. The linker should lay out sections without a priority number after those sections with.
- The priority values 0 to 100 inclusive are reserved to the implementation.
- Run-time support code iterates through the global constructor vector in increasing address order calling each identified initialization function in order.

#### 4.2.4 ELF binding of static data guard variable symbols

The generic C++ standard [GC++ABI] states at the end of §5.2.2:

Local static data objects generally have associated guard variables used to ensure that they are initialized only once (see 3.3.2). If the object is emitted using a COMDAT group, the guard variable must be too. It is suggested that it be emitted in the same COMDAT group as the associated data object, but it may be emitted in its own COMDAT group, identified by its name. In either case, it must be weak.

In effect the generic standard permits a producer to generate one of two alternative structures. Either:

```
COMDAT Group (Variable Name) {
    Defines Variable Name      // ELF binding STB_GLOBAL, mangled name
    Defines Guard Variable Name // ELF binding STB_WEAK, mangled name ...
}
```

Or:

```
COMDAT Group (Variable Name) {
    Defines Variable Name      // ELF binding STB_GLOBAL, mangled name
}
+
COMDAT Group (Guard Variable Name) {
    Defines Guard Variable Name // ELF binding STB_WEAK, mangled name
}
```

A link step involving multiple groups of the first kind causes no difficulties. A linker must retain only one copy of the group and there will be one definition of Variable Name and one weak definition of Guard Variable Name.

A link step involving pairs of groups of the second kind also causes no difficulties. A linker must retain one copy of each group so there will be one definition of Variable Name and one weak definition of Guard Variable Name.

A link step involving a group of the first kind and a pair of groups of the second kind generates two sub-cases.

- If the linker discards the group that defines two symbols there is no problem.
- If the linker retains the group that defines both Variable Name and Guard Variable Name it must nonetheless retain the group called Guard Variable Name. There are now two definitions of Guard Variable Name with ELF binding STB\_WEAK.

In this second case there is no problem provided the linker picks one of the definitions.

Unfortunately, [Generic ELF] does not specify how linkers must process multiple weak definitions when there is no non-weak definition to override them. If a linker faults duplicate weak definitions there will be a functional failure.

This ABI requires the ELF binding of Guard Variable Name in the first structure to be STB\_GLOBAL.

The rules codified in [Generic ELF] then make all three linking scenarios well defined and it becomes possible to link the output of compilers such as armcc that choose the first structure with the output of those such as gcc that choose the second without relying on linker behavior that the generic ELF standard leaves unspecified.

### 4.2.5 Unwind Table Location

Exception tables are located in sections with the name `.eh_frame` and `.eh_frame_hdr`. Linkers shall put the `.eh_frame_hdr` section in a single text segment, with a `PT_AARCH64_UNWIND` program table entry identifying the unwind table header location.



## EH ABI LEVEL III: IMPLEMENTATION ABI FOR GNU LINUX

### 5.1 Introduction

This section describes the Exception Handling Implementation ABI for GNU Linux systems.

It specifies:

- The format of the unwind tables
- Standard Runtime Initialization features
- Throwing an Exception
- Catching an Exception

This section follows the layout of [IA64EHABI<sup>5</sup>] §3.

### 5.2 Data Structures

The format of the exception tables is as specified in [LSB<sup>6</sup>] §II.11.6 (Exception Frames).

The codes used to describe the encoding of pointers used in the exception frame tables, are the values described in [LSB<sup>7</sup>] §II.11.5.1 (DWARF Exception Header Encoding).

Note that in particular that the layout of the Language Specific Data Area (LSDA) is not specified by this ABI. The structure and layout of a LSDA is specific to a particular implementation of a personality routine.

### 5.3 Standard Runtime Initialization

See [IA64EHABI<sup>8</sup>] §3.3.

### 5.4 Throwing an Exception

See [IA64EHABI<sup>9</sup>] §3.4.

---

<sup>5</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>

<sup>6</sup> [https://refspecs.linuxfoundation.org/LSB\\_4.0.0/LSB-Core-generic/LSB-Core-generic.html](https://refspecs.linuxfoundation.org/LSB_4.0.0/LSB-Core-generic/LSB-Core-generic.html)

<sup>7</sup> [https://refspecs.linuxfoundation.org/LSB\\_4.0.0/LSB-Core-generic/LSB-Core-generic.html](https://refspecs.linuxfoundation.org/LSB_4.0.0/LSB-Core-generic/LSB-Core-generic.html)

<sup>8</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>

<sup>9</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>

## 5.5 Catching an Exception

### 5.5.1 Overview of Catch Processing

Stack unwinding itself is begun by calling `__Unwind_RaiseException()`, and performed by the unwind library. See [IA64EHABI<sup>10</sup>] §3.5 for a summary.

### 5.5.2 The Personality Routine

The personality routine is specified in [IA64EHABI<sup>11</sup>] §2.5.2.

### 5.5.3 Exception Handlers

The behavior of exception handlers is described in [IA64EHABI<sup>12</sup>] §2.5.3.

---

<sup>10</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>

<sup>11</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>

<sup>12</sup> <https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html>