
Application Binary Interface for the Arm Architecture - the Base Standard

Release 2019Q4

Arm Ltd

Jan 31, 2020

CONTENTS

1	Preamble	2
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
2	About this document	5
2.1	Change control	5
2.2	References	5
2.3	Terms and abbreviations	6
2.4	Your licence to use this specification	7
2.5	Acknowledgements	8
3	Schematic map of the ABI for the Arm Architecture	9
3.1	Notes about the schematic map	10
4	The ABI for the Arm Architecture (base standard)	11
4.1	Overview and documentation map	11
4.2	Procedure call standard for the Arm architecture	12
4.3	C++ ABI for the Arm architecture	13
4.4	ELF for the Arm architecture	15
4.5	DWARF for the Arm architecture	16
4.6	Run-time ABI for the Arm architecture	17
4.7	The C library ABI for the Arm architecture	17
4.8	The base platform ABI for the Arm architecture	17
4.9	A note about ar format	18
4.10	Addenda to and errata in the ABI for the Arm Architecture	19

Document number: IHI 0036C, current through ABI release 2019Q4

Date of Issue: 30th January 2020

PREAMBLE

1.1 Abstract

This document describes the structure of the Application Binary Interface (ABI) for the Arm architecture, and links to the documents that define the base standard for the ABI for the Arm Architecture. The base standard governs inter-operation between independently generated binary files and sets standards common to Arm-based execution environments.

1.2 Keywords

ABI for the Arm architecture, ABI base standard, embedded 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 © 2003, 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

- *Application Binary Interface for the Arm® Architecture – The Base Standard* (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)
 - *Current status and anticipated changes* (page 5)
 - *Change history* (page 5)
 - * *References* (page 5)
 - * *Terms and abbreviations* (page 6)
 - * *Your licence to use this specification* (page 7)
 - * *Acknowledgements* (page 8)
- *Schematic map of the ABI for the Arm Architecture* (page 9)
 - * *Notes about the schematic map* (page 10)
- *The ABI for the Arm Architecture (base standard)* (page 11)
 - * *Overview and documentation map* (page 11)
 - * *Procedure call standard for the Arm architecture* (page 12)
 - * *C++ ABI for the Arm architecture* (page 13)
 - *The Generic C++ ABI* (page 13)
 - *The C++ ABI supplement for the Arm architecture* (page 14)
 - *The Exception handling ABI for the Arm architecture* (page 14)
 - *The exception handling components specimen implementation* (page 14)
 - * *ELF for the Arm architecture* (page 15)
 - *The generic ELF specification* (page 15)
 - *ELF for the Arm architecture (processor supplement)* (page 15)
 - * *DWARF for the Arm architecture* (page 16)
 - *DWARF 3.0* (page 16)
 - *ABI DWARF usage conventions* (page 16)
 - * *Run-time ABI for the Arm architecture* (page 17)
 - * *The C library ABI for the Arm architecture* (page 17)
 - * *The base platform ABI for the Arm architecture* (page 17)
 - * *A note about **ar** format* (page 18)
 - * *Addenda to and errata in the ABI for the Arm Architecture* (page 19)
 - *Build attributes* (page 19)
 - *Thread local storage* (page 19)

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.

2.1.2 Change history

Issue	Date	By	Change
1.0	30 th October 2003	LS	First public release.
2.0	24 th March 2005	LS	Second public release.
A	24 th October 2007	LS	Document renumbered (formerly GEN-003535 v2.0).
B	10 th October 2008	LS	A note about <i>ar</i> format (page 18) fixed a typo and updated the reference to ar format.
2018Q4	21 st December 2018	OS	Minor typographical fixes, updated links.
2019Q4	30 th January 2020	TS	Minor layout changes.

2.2 References

This document refers to the following documents.

Ref	External URL	Title
AADWARF		DWARF for the Arm Architecture
AAELF		ELF for the Arm Architecture
AAPCS		Procedure Call Standard for the Arm Architecture
ADDENDA		Addenda to, and errata in, the ABI for the Arm Architecture
BPABI		Base Platform ABI for the Arm Architecture
<i>BSABI</i> (page 1)	This standard	ABI for the Arm Architecture (Base Standard)
CLIBABI		C Library ABI for the Arm Architecture
CPPABI		C++ ABI for the Arm Architecture
EHABI		Exception Handling ABI for the Arm Architecture
EHEGI		Exception handling, components, example implementations
RTABI		Run-time ABI for the Arm Architecture
GCPPABI ¹	https://github.com/itanium-cxx-abi/cxx-abi	Generic C++ ABI
GDWARF ²	http://dwarfstd.org/Dwarf3Std.php	DWARF 3.0, the generic debug table format.
GABI ³	http://www.sco.com/developers/gabi/	Generic ELF, 17 th December 2003 draft.
GLSB ⁴	http://refspecs.linuxfoundation.org/lsb.shtml	gLSB v1.2 Linux Standard Base
OpenBSD ⁵	http://www.openbsd.org/	Open BSD Standard

2.3 Terms and abbreviations

The *ABI for the Arm Architecture* uses the following terms and abbreviations.

AAPCS Procedure Call Standard for the Arm 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, the Run-time ABI for the Arm Architecture, the C Library ABI for the Arm Architecture.

AEABI (Embedded) ABI for the Arm architecture (this ABI...)

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

core registers The general purpose registers visible in the Arm architecture's programmer's model, typically r0-r12, SP, LR, PC, and CPSR.

EABI An ABI suited to the needs of embedded, and deeply embedded (sometimes called free standing), applications.

¹ <http://itanium-cxx-abi.github.io/cxx-abi/abi.html>

² <http://dwarfstd.org/Dwarf3Std.php>

³ <http://www.sco.com/developers/gabi/>

⁴ <http://refspecs.linuxfoundation.org/lsb.shtml>

⁵ <http://www.openbsd.org/>

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.

VFP The Arm architecture’s Floating Point architecture and instruction set. In this ABI, this abbreviation includes all floating point variants regardless of whether or not vector (V) mode is supported.

2.4 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)

2.5 Acknowledgements

This specification has been developed with the active support of the following organizations. In alphabetical order: Arm, CodeSourcery, Intel, Metrowerks, Montavista, Nexus Electronics, PalmSource, Symbian, Texas Instruments, and Wind River.

SCHEMATIC MAP OF THE ABI FOR THE ARM ARCHITECTURE

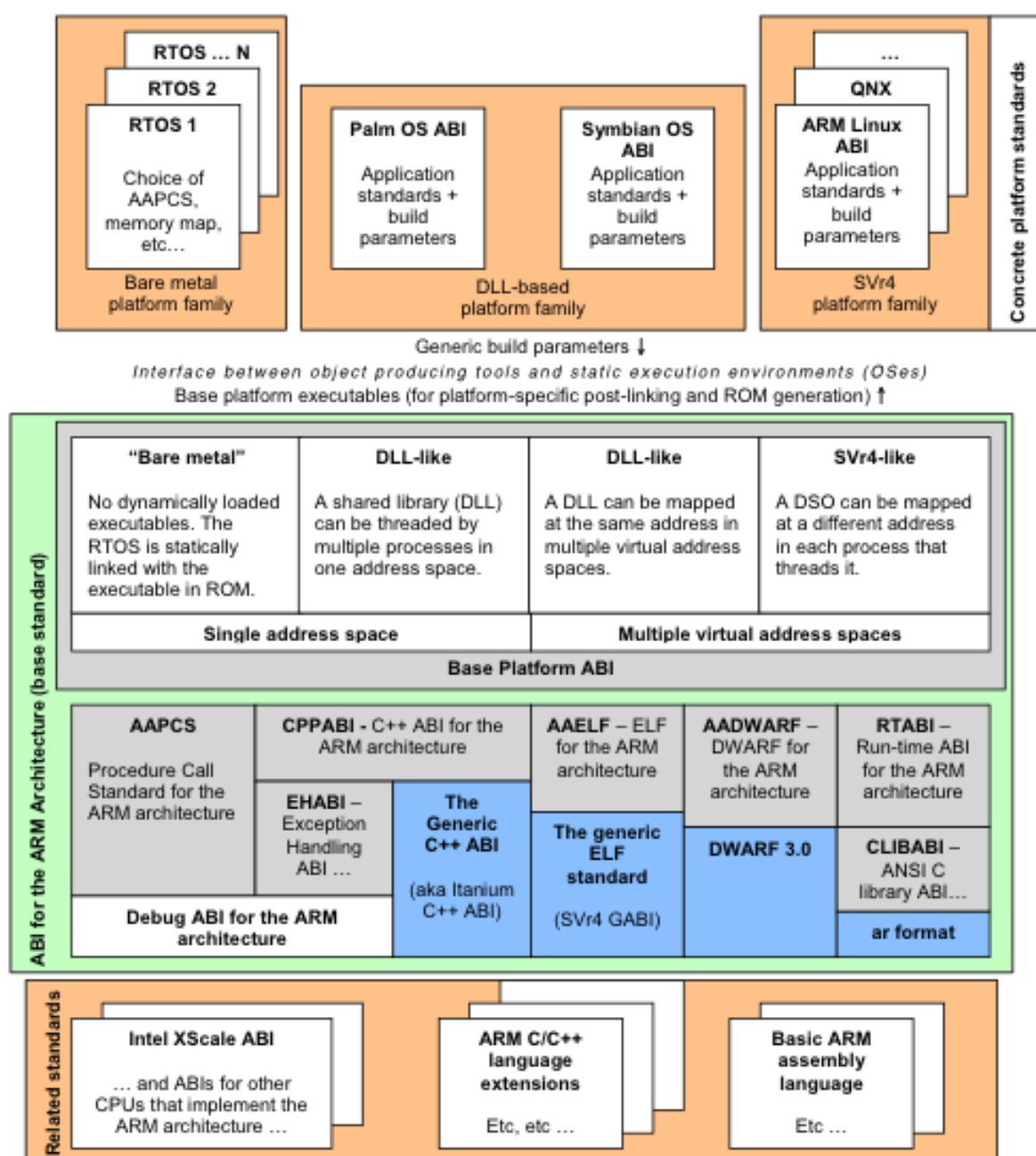


Fig. 3.1: A schematic map of the ABI for the Arm Architecture and some related standards

3.1 Notes about the schematic map

Pale gray boxes depict the most important components of the base standard for the *ABI for the Arm Architecture*.

Pastel blue (or darker gray on a gray-scale printed copy) boxes depict the most important external standards we refer to. We do not show them all – for example, we also refer to the ANSI standards for programming languages C and C++ and to the IEEE 754 standard for floating-point arithmetic.

The tan (also darker gray on a gray-scale printed copy) annotation boxes label groups of related standards that might be developed in the future, and a pastel green box (pale gray on a gray-scale printed copy) encloses all components (direct and referenced) of the *ABI for the Arm Architecture* (base standard).

The size of each box is unrelated to the size or significance of the component depicted.

Sections depicted with white boxes on a tan background are beyond the scope of *this* base standard. In each case the section involves either or both of the following.

- A third party on whom there is no obligation to contribute.
- Future intentions to which there is no current commitment.

The sections depicted with white boxes on a tan background show the position of *this base standard* in a larger context. They depict some of the ways in which those affected by this ABI standard might like to grow it, and how the base standard would relate to other plausible pieces of a larger jigsaw of Arm architecture-related standards. In no case shall this depiction be interpreted as an intention or commitment by Arm or any third party to create the component standard depicted.

The ABI for the Arm Architecture (base standard) (page 11), below, describes the base standard in detail and refers to each of its components.

THE ABI FOR THE ARM ARCHITECTURE (BASE STANDARD)

4.1 Overview and documentation map

The *ABI for the Arm Architecture* is a collection of standards, some open and some specific to the Arm architecture, that regulate the inter-operation of binary files and development tools in a spectrum of Arm-based execution environments from bare metal to major operating systems such as Arm Linux. We expect that ABIs for specific execution environments will build on, and extend, the slices of this ABI that apply to them.

Standardizing the inter-operation of binary files requires standardizing certain aspects of code generation itself, so this base standard is aimed principally at the authors and vendors of C and C++ compilers, linkers, and run-time libraries. In general, there can be no complying executable files until there are complying relocatable files.

Table 4.1: Table 1, Documentation map of the ABI for the Arm architecture base standard

Component standard	Base standard
The <i>Procedure Call Standard for the the Arm Architecture</i> [AAPCS], is summarized in <i>Procedure call standard for the Arm architecture</i> (page 12).	None.
The C++ ABI for the Arm Architecture [CPPABI] is summarized in <i>C++ ABI for the Arm architecture</i> (page 13). It details where the C++ ABI for the ABI deviates from the base standard.	The Generic C++ ABI (<i>aka C++ ABI for Itanium</i>). https://github.com/itanium-cxx-abi/cxx-abi
The <i>Exception Handling ABI for the Arm Architecture</i> [EHABI], is summarized in <i>The Exception handling ABI for the Arm architecture</i> (page 14). It describes C++-specific and language-independent exception processing.	
<i>ELF for the Arm Architecture</i> [AAELF] is summarized in <i>ELF for the Arm architecture</i> (page 15). It gives processor-specific and platform-specific details not given in the generic ELF specification.	The generic ELF standard (SVr4 GABI), 17 th December 2003 draft. http://www.sco.com/developers/gabi/
<i>DWARF for the Arm Architecture</i> [AADWARF] is summarized in <i>DWARF for the Arm architecture</i> (page 16). It describes how DWARF should be used to promote inter-operation between independent producers and consumers.	DWARF 3.0. http://dwarfstd.org/Dwarf3Std.php
The <i>Run-time ABI for the Arm Architecture</i> [RTABI] is summarized in <i>Run-time ABI for the Arm architecture</i> (page 17). It specified a helper-function ABI to support C, C++, and arithmetic (floating-point, integer division, and non-trivial long long arithmetic).	The unix ar format is the base standard for libraries of relocatable ELF files (see <i>A note about ar format</i> (page 18)).
The <i>C Library ABI for the Arm Architecture</i> [CLIBABI], is summarized in <i>The C library ABI for the Arm architecture</i> (page 17). It describes an ANSI C library ABI that can easily be supported by existing libraries.	ISO/IEC 9899:1990 Programming languages - C, with some reference to ISO/IEC 9899:1999. See also <i>A note about ar format</i> (page 18) re ar format
The <i>Base Platform ABI for the Arm Architecture</i> [BPABI], is summarized in <i>The base platform ABI for the Arm architecture</i> (page 17). It specified executable and shared object files suited to the execution environments supported by this ABI, and the static linker functionality required to create them.	The generic ELF standard (SVr4 GABI), 17 th December 2003 draft. http://www.sco.com/developers/gabi/ Linux Standard Base v1.2 specification [GLSB ⁶]
<i>Addenda to, and errata in, the ABI for the Arm Architecture</i> [ADDENDA] contains late additions to this version of the ABI specification, summarized in <i>Addenda to and errata in the ABI for the Arm Architecture</i> (page 19).	None.

The ABI for the Arm architecture base standard comprises the component standards listed in *Table 1, Documentation map of the ABI for the Arm architecture base standard* (page 12). The scope and purpose of each component is explained in following subsections referred to from the table.

4.2 Procedure call standard for the Arm architecture

The Procedure Call Standard for the Arm architecture [AAPCS] specifies:

- The size, alignment, and layout of C and C++ *Plain Old Data* (POD) types including
 - Primitive data types.
 - Structures.

⁶ <http://refspecs.linuxfoundation.org/lsb.shtml>

- Enumerated types.
 - Bit field types.
 - Primitive types specific to C++ (references and pointers to members).
 - How to pass control and data between publicly visible functions. A function is publicly visible if its callers are translated separately from it, and some callers might have no knowledge of how it was translated, other than that it conforms to the AAPCS.
- (When the public visibility of F is made explicit – for example by using a `#pragma` or annotation such as `__export` or `__declspec(dllexport)` – we also describe F as *exported*).
- Use of the run-time stack, and the stack invariants that must be preserved.

4.3 C++ ABI for the Arm architecture

The C++ ABI for the Arm architecture comprises four sub-components.

- The generic C++ ABI, summarized in *The Generic C++ ABI* (page 13), is the referenced base standard for this component.
- The C++ ABI supplement for the Arm architecture, summarized in *The C++ ABI supplement for the Arm architecture* (page 14), details Arm-specific deviations from the generic standard and records Arm-specific interpretations of it.
- The separately documented *Exception Handling ABI for the Arm Architecture*, summarized in *The Exception handling ABI for the Arm architecture* (page 14), describes the language-independent and C++-specific aspects of exception handling.
- The specimen implementations of the exception handling components, summarized in *The exception handling components specimen implementation* (page 14), include:
 - A language independent unwinder.
 - A C++ semantics module.
 - Arm-specific C++ personality routines.

4.3.1 The Generic C++ ABI

The generic C++ ABI (originally developed for Itanium, [GCPPABI⁷]) 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*, summarized in *Procedure call standard for the Arm architecture* (page 12)).
- 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 represented as ELF symbols (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 *Exception Handling ABI for the Arm Architecture* (*The Exception handling ABI for the Arm architecture* (page 14)).

⁷ <http://itanium-cxx-abi.github.io/cxx-abi/abi.html>

4.3.2 The C++ ABI supplement for the Arm architecture

The Arm C++ ABI supplement is a major section in the document *C++ ABI for the Arm Architecture* [CPPABI].

The Arm C++ ABI supplement describes where the C++ ABI for the Arm architecture necessarily diverges from the generic C++ ABI, because Itanium-specifics that cannot work (efficiently) for the Arm architecture show through an otherwise generic specification. For example, the generic encoding of a pointer to member function uses the least significant bit of a word to distinguish a code address from a v-table offset. The Arm architecture uses the same bit to distinguish Arm-code from Thumb-code, so the Arm ABI must deviate.

4.3.3 The Exception handling ABI for the Arm architecture

In common with the Itanium exception handling ABI, the *Exception Handling ABI for the Arm architecture* [EHABI] specifies table-based stack unwinding that separates language-independent unwinding from language specific concerns. The Arm specification describes:

- The *base class* understood by the language-independent exception handling system, and its representation in object files. The language-independent exception handler only uses fields from this base class.
- A *derived class* used by Arm tools that efficiently encodes stack-unwinding instructions and compactly represents the data tables needed for handling C++ exceptions.
- The interface between the language-independent exception handling system and the *personality routines* specific to a particular implementation for a particular language. Personality routines interpret the language specific, derived class tables. Conceptually (though not literally, for reasons of implementation convenience and run-time efficiency), personality routines are member functions of the derived class.
- The interfaces between the (C++) language exception handling semantics module and
 - The language independent exception handling system.
 - The personality routines.
 - The (C++) application code (effectively the interface underlying throw).

The *Exception Handling ABI for the Arm Architecture* contains a significant amount of commentary to aid and support independent implementation of:

- Personality routines.
- The language-specific exception handling semantics module.
- Language independent exception handling.

This commentary does not provide, and is not intended to provide, a complete guide to independent implementation, but it does give a rationale for the interfaces to, and among, these components.

4.3.4 The exception handling components specimen implementation

Licence to use the exception handling components specimen implementation

The licence to use the specimen implementation of the exception handling components is included in the zip file containing them (as the file `LICENCE.txt`) and referred to from each source file. It is broadly similar in scope and intent to the licence to use this specification displayed in *Your licence to use this specification* (page 7).

Contents of the exception handling components example implementation

The exception handling components example implementation [EHEGI] comprises the following files.

- `cppsemantics.cpp` is a module that implements the semantics of C++ exception handling. It uses the language-independent unwinder (`unwinder.c`), and is used by the Arm-specific personality routines (`unwind_pr.[ch]`).
- `cxxabi.h` describes the generic C++ ABI (*The Generic C++ ABI* (page 13)).
- `LICENCE.txt` contains your licence to use, copy, modify, and sublicense the specimen implementation.
- `unwind_env.h` is a header that describes the build and execution environments of the exception handling components. This header must be edited if the exception handling components are to be built with non-Arm compilers. This header `#includes cxxabi.h`.
- `unwind_pr.c` implements the three Arm-specific personality routines described in the *Exception Handling ABI for the Arm Architecture*.
- `unwinder.c` is an implementation of the language-independent unwinder.
- `unwinder.h` describes the interface to the language-independent unwinder, as described in the *Exception Handling ABI for the Arm Architecture*.

4.4 ELF for the Arm architecture

ELF for the Arm architecture comprises two components.

- The generic ELF specification, summarized in *The generic ELF specification* (page 15).
- The ELF processor supplement for the Arm architecture, summarized in *ELF for the Arm architecture (processor supplement)* (page 15).

4.4.1 The generic ELF specification

The generic *Executable and Linking Format* specification was originally developed for Unix System V by AT&T. The latest version and the most recent stable drafts are published by *The SCO Group* at [GABI⁸]. They specify:

- The format and meaning of statically linkable object files.
- The format and meaning of executable and shared-object files.

In each case, a supplement specifies processor-specific and platform-specific aspects.

- The enumeration of relocation directives is specific to a processor. Often, this is the only processor-specific facet of statically linkable (relocatable) ELF files.
- For executable files a platform-specific supplement specifies the interface to loading and dynamic linking.

4.4.2 ELF for the Arm architecture (processor supplement)

ELF for the Arm Architecture [AAELF] describes the following:

- The representation in ELF and generation of cross-platform executable file information required by the *Base Platform ABI for the Arm Architecture* (*The base platform ABI for the Arm architecture* (page 17) and [BPABI]).
- Symbol versioning information.

⁸ <http://www.sco.com/developers/gabi/>

- Symbol pre-emption information.
- Procedure linkage table (PLT) entries, also known to users of the Arm architecture as intra-call veneers.
- The enumeration of static and dynamic relocation directives.
- Processor-specific flags and conventions (for example, the *Mapping symbols* described in aael32-section4-5-5 of [AAELF], used to accommodate the use of the Arm and Thumb instruction sets in the same code section).
- Two kinds of big-endian executable file (corresponding to the two flavors of big-endian code defined by Arm architecture v6 – in a BE8 big-endian executable file, code is nonetheless encoded little-endian).
- Miscellaneous Arm-specific executable and shared-object flags and section types used by the *ABI for the Arm Architecture*.

The Base Platform ABI for the Arm Architecture (*The base platform ABI for the Arm architecture* (page 17) and [BPABI]) specifies how ELF is used to support the executable file organizations and execution environments depicted in *A schematic map of the ABI for the Arm Architecture and some related standards* (page 9).

4.5 DWARF for the Arm architecture

DWARF for the Arm architecture comprises two components.

- The generic DWARF specification, DWARF 3.0, summarized in *DWARF 3.0* (page 16).
- The Arm DWARF usage conventions, summarized in *ABI DWARF usage conventions* (page 16).

4.5.1 DWARF 3.0

DWARF 3.0 [GDWARF⁹] makes precise many ambiguous and ill-defined aspects of the DWARF 2.0 specification, and extends that specification with:

- Additional constructs for describing optimized code and stack unwinding.
- Additional constructs for describing C++, Java, and Fortran 90.

4.5.2 ABI DWARF usage conventions

The ABI DWARF usage conventions are described in aadwarf32-section3 of the document *DWARF for the Arm Architecture* [AADWARF]. This section defines:

- An Arm-specific allocation of DWARF register numbers (in `.debug_frame` unwind descriptions).
- How Arm-state and Thumb-state are encoded in DWARF line number tables.
- How to describe data known to be in the other byte order (Arm architecture v6 access to other-endian data).
- The Canonical Frame Address (CFA).
- The default interpretation of debug frame Common Information Entries (CIEs).

⁹ <http://dwarfstd.org/Dwarf3Std.php>

4.6 Run-time ABI for the Arm architecture

The run-time helper-function ABI is described in the document *Run-time ABI for the Arm Architecture* [RTABI].

The run-time helper-function ABI specifies how relocatable files produced by one tool chain must inter-operate with the run-time library from a different tool chain or execution environment. It gives a simple model of what a producer may assume of its output's eventual static linking and execution environments. It defines the following.

- A minimum model of floating-point arithmetic, based on the IEEE 754 floating-point arithmetic standard:
 - To which producers of relocatable files must conform.
 - Which producers of relocatable files can assume of the eventual execution environment.

(The model sets a minimum standard. Implementers may implement the full IEEE 754 specification).

- The type signatures, meaning, and allowable names of the helper functions that all conforming static linking environments must support. The set of helper functions is divided into those required by C and assembly language, and those required only by C++.
- The provision, as part of the relocatable object itself or in separately delivered libraries, of all other helper functions used by a translation unit.

Libraries of relocatable ELF files must be formatted as Unix-style **ar** format linkable libraries (see [A note about ar format](#) (page 18), below).

4.7 The C library ABI for the Arm architecture

The C library ABI is described in the document *C Library ABI for the Arm Architecture* [CLIBABI].

The C library ABI specifies:

- A binary interface to the C89 run-time library that allows a C-library-using function built by one tool chain to use the C library implementation provided by another.
- Constraints on language library headers necessary to allow tool chain X to use its own headers, or tool chain Y's headers, when building an object that must interface to tool chain Y's run-time library.

Compliance with this specification is a header-by-header *quality of implementation* issue. Compliance is not required in order to claim compliance to this base standard ABI for the Arm architecture.

Libraries of relocatable ELF files must be formatted as Unix-style **ar** format linkable libraries (see [A note about ar format](#) (page 18), below).

4.8 The base platform ABI for the Arm architecture

The base platform ABI is described in the document *Base Platform ABI for the Arm Architecture* [BPABI].

The base platform ABI specifies:

- The content and format of ELF-based executable files suitable for post-processing to platform-specific binary formats appropriate to the families of execution environment supported by this ABI ([A schematic map of the ABI for the Arm Architecture and some related standards](#) (page 9)).
- The division of responsibility between static linkers generating fully symbolic executable ELF files and post-linkers generating less symbolic, platform-specific executable files.

- The static linking functionality needed to generate a generic executable file – the functionality needed to encompass the platform families supported by this ABI.

In most cases, some platform-specific post-processing is required to produce a platform executable file, but the complexity of the post processor is limited:

- For the SVr4 platform family, the required post-processing is a tiny increment on the static linking needed to generate a BPABI executable file. We expect most static linkers will offer an option to directly generate an executable file for Linux.
- For the DLL-based platform families platform-specific post-linking is significant, but little more complicated than an off-line version of SVr4 dynamic linking followed by a file format conversion.
- The bare metal platform family may demand additional static linking functionality to manage separate load and execution addresses and multiple image segments. Extracting such segments from an ELF executable file to drive ROM generating tools is trivial in comparison with the above tasks.

We expect post linking to be used primary in support of DLL-based platforms and specialized execution environments that feature dynamically loaded executable files.

4.9 A note about ar format

This ABI specifies that libraries of relocatable ELF files must be formatted as Unix-style **ar** format linkable libraries. This section specifies the **ar** variant used by Arm tools.

Unfortunately, **ar** format is not well standardized, and good public references to the format are hard to find. The **ar** command is deprecated from the Linux base standard [GLSB¹⁰] which states that it is “... expected to disappear from a future version of the LSB”.

A good general introduction to **ar** format, including a brief history and a warning about the incompatibility of its variants, is given in the *Manuals* section of [OpenBSD¹¹]. Search there for **ar** in section 5 – *File Formats*. However, please be aware of the following concerning the name field in archive headers:

- Different **ar** variants manage long file names (> 14 characters), and file names containing spaces, differently.
- *RealView* tools from Arm do *not* use the BSD file name conventions described at [OpenBSD¹²].

Recently, we have found a *Wikipedia* article about **ar** format [[http://en.wikipedia.org/wiki/Ar_\(Unix\)](http://en.wikipedia.org/wiki/Ar_(Unix))]. The *GNU variant* it describes is similar to the *RealView* variant summarized immediately below with this difference.

- As of early October 2008, this *Wikipedia* article claims that the 32-bit binary integers in the symbol table member (called ‘/’) are encoded *big endian*.
- Arm targeted GNU tools and *RealView* tools always encode binary data using the byte order of the target system – little endian for little endian targets and big endian for big endian targets.

ar format conventions used by RealView tools and Arm-targeted GNU tools

File names recorded in archive member headers are terminated with a ‘/’. This allows short (≤ 14 characters) names to contain spaces.

The symbol table member (always present if an archive contains relocatable files) has the header name ‘/’. The symbol table member contains, in order:

- A 32-bit count of the number of symbols in the table. The byte order is that of the target system.

¹⁰ <http://refspecs.linuxfoundation.org/lsb.shtml>

¹¹ <http://www.openbsd.org/>

¹² <http://www.openbsd.org/>

- For each symbol, the 32-bit offset within the archive of the header of the member defining it. The byte order is that of the target system.
- The NUL-terminated name of each symbol, listed in the same order as the offsets.

There is always a file names member with the header name `''`. It contains the names of all the files in the archive. Each name is terminated by `'/'` followed by `'n'` (so the member contains only printable text).

If the file name of an archive member is longer than 14 characters, its header name is `'/'` followed by the decimal offset of its name in the file names member. Otherwise the header name is the file name of the member.

Ordinary members follow the symbol table member and the file names member.

4.10 Addenda to and errata in the ABI for the Arm Architecture

Addenda to, and errata in, the ABI for the Arm Architecture [ADDENDA] contains late additions to version 2.0 (*this version*) and will contain any significant additions made during future maintenance of v2.0.

As of the publication of v2.0 of the *ABI for the Arm Architecture* (date shown [at the top](#) (page 1) of this document), there are two addenda, addenda32-section2 and addenda32-section3.

As of this publication date (shown [at the top](#) (page 1) of this document) there are no errata.

4.10.1 Build attributes

Build attributes record:

- The use of architectural features and ABI variants by the code and data in a relocatable file.
- To a limited extent, the intentions of the builder of the file.

Attributes allow linkers to determine whether separately built relocatable files are inter-operable or incompatible, and to select the variant of a required library member that best matches the intentions of their builders.

4.10.2 Thread local storage

Thread Local Storage (TLS) is a class of *own data* (static storage) that – like the stack – is instanced once for each thread of execution.

This addendum defines the thread local storage (TLS) model for Linux for the Arm architecture. It covers:

- An introduction to the ABI issues raised by thread local storage.
- An introduction to addressing thread local variables.
- How Linux for the Arm architecture addresses thread local variables:
 - How thread local variables must be addressed from dynamically loadable DSOs.
 - How thread local variables may be addressed more efficiently from applications and DSOs loaded only when a process is created.

The Linux-specific TLS relocations are described in [AAELF] (*ELF for the Arm architecture* (page 15)).