Linaro connect
Budapest 2017

# BUD17 - 308
# Navigating the ABI for the ARM Architecture

Peter Smith

# Agenda

- Introduction to the ABI, and its history
- The structure of the ABI and how it fits together with other standards
- Expectations of compatibility
- The 64-bit ABI and how it differs from the 32-bit

# My background with the ABI

- Worked in the proprietary ARM Compiler toolchain from 2000 - 2016
  - ADS 1.0, RVCT, ARM Compiler 5, ARM Compiler 6
  - Main focus is on embedded systems
  - Limited intersection with ARM Linux
- Specializing in non-compiler tools, such as the linker, assembler and object processing tools
  - Armlink, armasm, fromelf
- Involved in implementing, rather than specifying the ABI

# Definitions

- **A**pplication **P**rogramming **I**nterface **API**
  - Interface at the source code level
- **A**pplication **B**inary **I**nterface **ABI**
  - Interface between executables, shared libraries and operating systems
- **E**mbedded **A**pplication **B**inary **I**nterface
  - Interface between relocatable objects
- Platform
  - A software platform running a sophisticated OS that can run applications
  - Examples include Linux, *BSD, Symbian
- Bare Metal
  - An embedded application running without an OS, or at most an RTOS
- Quality **o**f **I**mplementation **Q-o-I**
  - Additional functionality over and above the minimum required for conformance, or permitted collusion between components made by same vendor

# What is the ABI?

- What does an ABI define and why should I care?
- ARM in early 2000s and the influence on the ABI
- Motivations and principles behind the ABI

# What does an ABI define?

- Procedure calling standard
- Sizes, layout and alignment of types
- C++ name mangling
- Exception handling
- Object file and library file format
- Debug information format

Mandatory for a C/C++ ABI

- Thread local storage?
- Compiler Helper Functions and runtime library?
- Dynamic Linking?
- System call interface?

Depends on scope of ABI

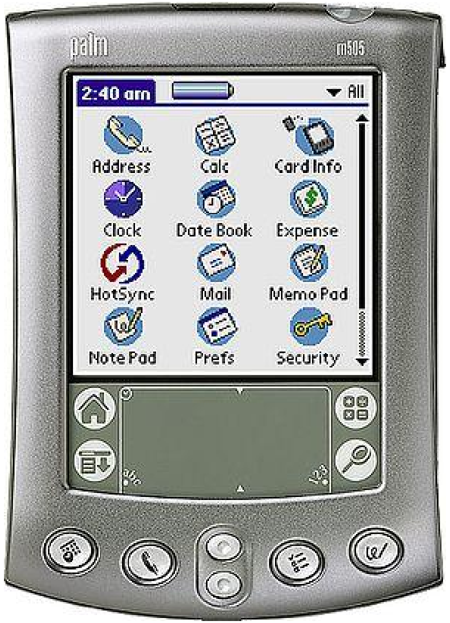
# Why should I care about the ABI?

- Majority of the ABI hidden from you by development tools
- Necessary if you are implementing development tools
  - Not just large tools like compiler, linker, assembler and debugger
  - Custom object file processors
- Developing or distributing cross platform binaries
  - What level of interoperability can you expect between toolchains?
- Understanding the different calling conventions available

# ARM in early 2000s

- The ABI for the ARM Architecture was released on 30th October 2003
- ARM11 family (v6) released to partners late 2002, ARM7 and ARM9 popular
- ARM’s largest market by far was Mobile with custom ASICs
- Constrained devices, with expensive flash prices
- Software mostly embedded, device specific and not upgradeable
    - SW Could be targeted at, and optimized for a specific device
- Fragmented market in tools and operating systems
    - Upwards of 20 available toolchains
- Early signs of commercial interest in Linux
    - Consumer Electronics Linux Forum founded in June 2003

# ARM consumer products from early 2000s



# ABI motivation and goals

- IA64 C++ ABI recently available on both GCC and EDG
- Linux on ARM showing signs of gathering momentum
- RTOS vendors needing to ship a different binary package for every toolchain
- Enable independent development tool chains to support inter-operation between portable binary packages
    - Use any application library in any ARM environment
    - Use any object producer with any ARM-based platform

# ABI for the ARM Architecture Principles

- Must be read in conjunction with other documents such as the **ARM ARM**
- ABI builds upon industry standards such as **ELF** and **Dwarf**
- Platform owners expected to define their own **ABI** when required
- Conformance follows the **“as if”** rule, if a conforming external observer cannot detect non-conformance there is no need to conform
- Tools vendors must be free to differentiate on Q-o-I even at ‘extern’ interfaces
  - Components have an exported interface with ‘contractual’ guarantees of conformance
  - An exported interface is ‘extern’ but not all ‘extern’ interfaces are exported
- Multiple options when consensus not reachable

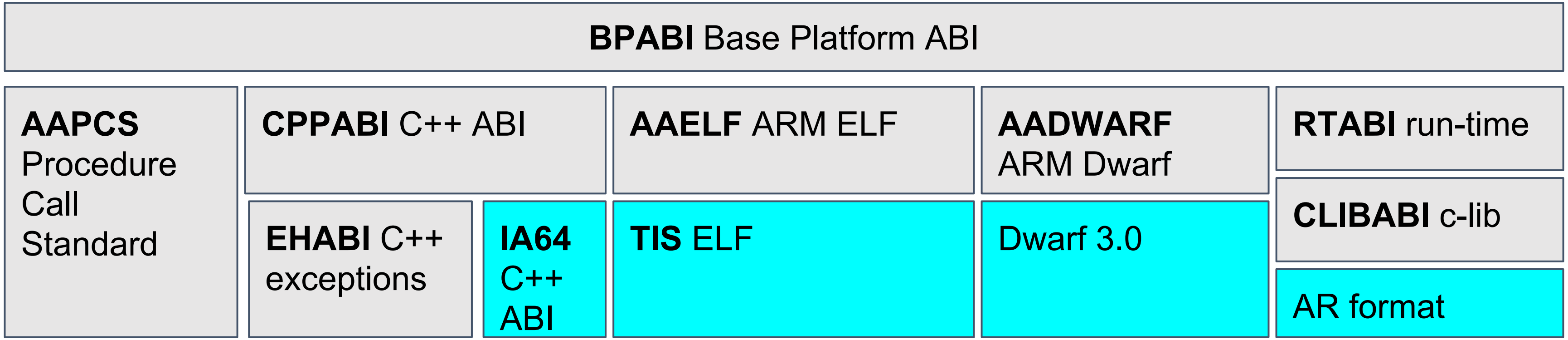
# ABI Structure

- The ABI documents and how they relate to generic and platform standards

# ABI for the ARM Architecture structure

Instead of a System V ABI ARM Processor Supplement, the ABI is split up into several documents

- ABI documents in gray
- Generic industry documents in cyan
- BPABI is the interface to platforms



**BPABI** Base Platform ABI

**AAPCS** Procedure Call Standard

**CPPABI** C++ ABI

**EHABI** C++ exceptions

**IA64** C++ ABI

**AAELF** ARM ELF

**TIS** ELF

**AADWARF** ARM Dwarf

Dwarf 3.0

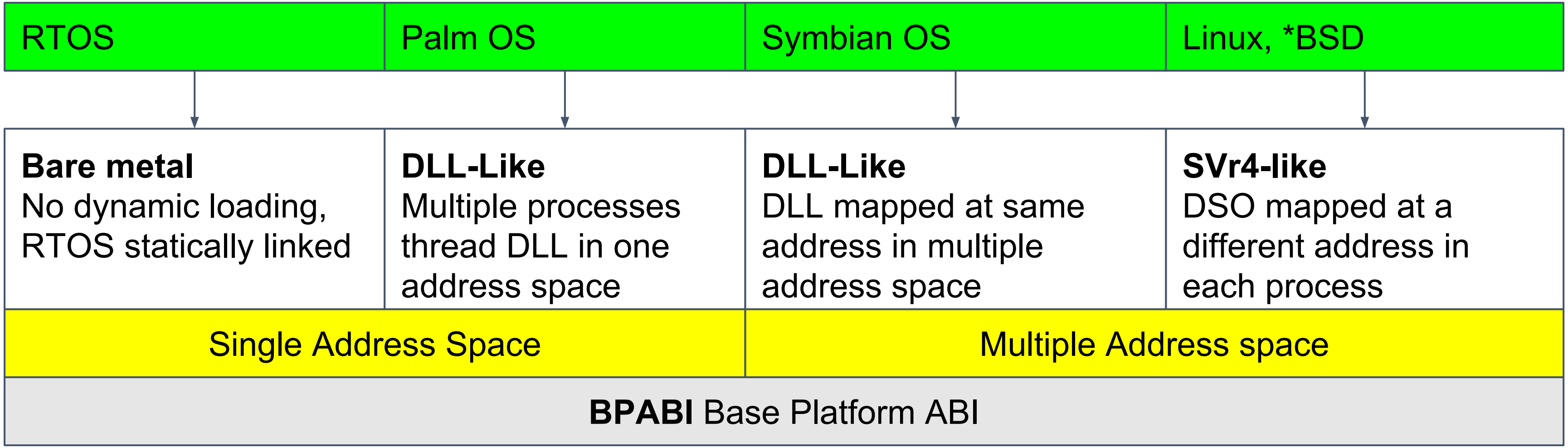
**RTABI** run-time

**CLIBABI** c-lib

AR format

# ABI for the ARM Architecture structure

4 types of platform are supported by the base platform ABI, with an example of each one.



RTOS

Palm OS

Symbian OS

Linux, *BSD

**Bare metal**
No dynamic loading, RTOS statically linked

**DLL-Like**
Multiple processes thread DLL in one address space

**DLL-Like**
DLL mapped at same address in multiple address space

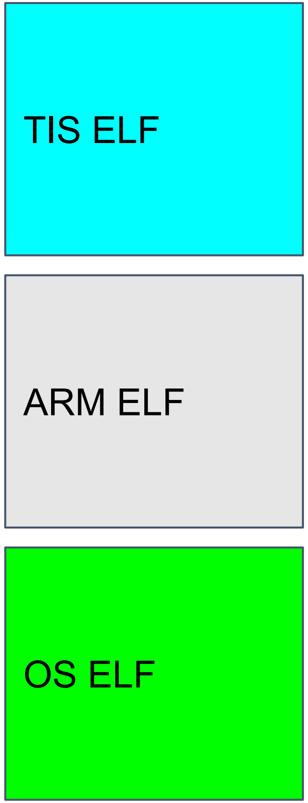
**SVr4-like**
DSO mapped at a different address in each process

Single Address Space

Multiple Address space

**BPABI** Base Platform ABI

# Layering of Standards, ELF example



TIS ELF

ARM ELF

OS ELF

Generic

- Concepts common to all uses of ELF
- Extension points for processor and platform (OS)

Often defined by the SystemV ABI document for the architecture

Processor Specific

- Concepts specific to ARM’s interpretation of ELF
- Relocation directives
- Flags and types in the processor specified range

Platform Specific

- Concepts specific to a platform’s interpretation of ELF
- Dynamic Relocation directives, TLS Model
- Flags and types in the OS specified range

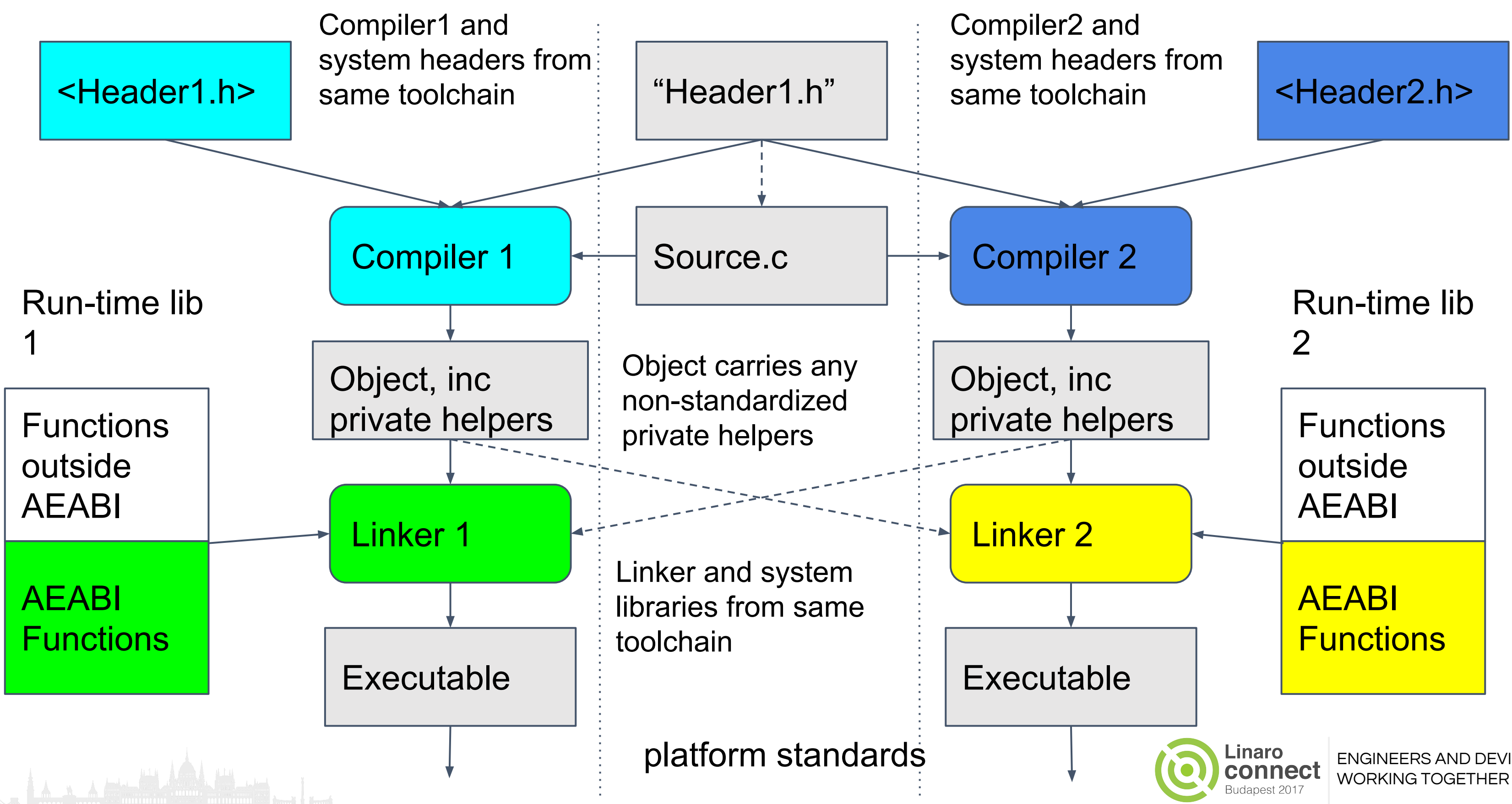
# ABI Documents

- The core of the ABI is related to relocatable object compatibility
  - ARM Procedure call standard
    - Includes base standard and variants such as VFP
  - ELF for the ARM Architecture, ARM processor specific supplement psABI
    - Includes relocations for all platforms
  - Dwarf for the ARM Architecture
    - mapping of register numbers
  - Exception handling ABI for the ARM Architecture
    - Table based but specific to ARM
  - C++ ABI for the ARM Architecture
    - Deviations from the Itanium C++ ABI standard
  - Run time ABI for the ARM Architecture
    - Compiler helper functions and built-ins
  - C-library ABI for the ARM Architecture
    - Compatibility model for C-libray interoperation
  - Errata and Addenda to the ARM Architecture (TLS and build attributes)

# Relocatable Object Compatibility

- What can I expect for C and C++ objects produced by different toolchains?
- How to check binary object properties?

# C-Library compatibility model



<Header1.h>
Compiler1 and system headers from same toolchain
"Header1.h"
Compiler2 and system headers from same toolchain
<Header2.h>
Compiler 1
Source.c
Compiler 2
Run-time lib 1
Run-time lib 2
Object, inc private helpers
Object carries any non-standardized private helpers
Object, inc private helpers
Functions outside AEABI
AEABI Functions
Linker 1
Linker 2
Functions outside AEABI
AEABI Functions
Linker and system libraries from same toolchain
Executable
Executable
platform standards

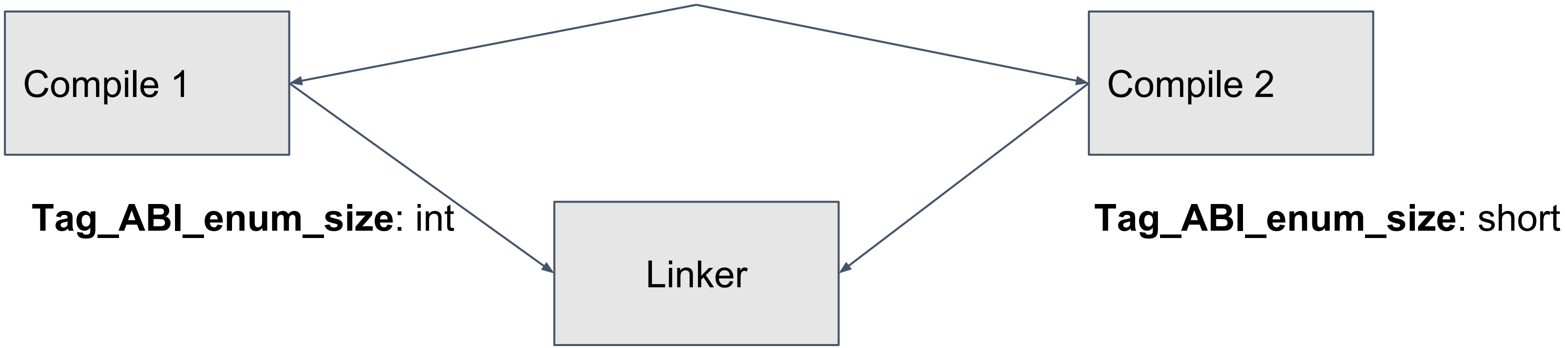
# Q-o-I Managing compatibility between objects

- Build Attributes capture the intention to use short or int enums, a potentially incompatible choice if each side of the API chooses differently.
- Linker can check for clashes in the attributes

```
typedef enum {val1 = 0, val2 = 1} Vals;
extern void api_function(Vals* v);
static void internal_function(Vals* v) { … }
```

-fno-short-enums

-fshort-enums

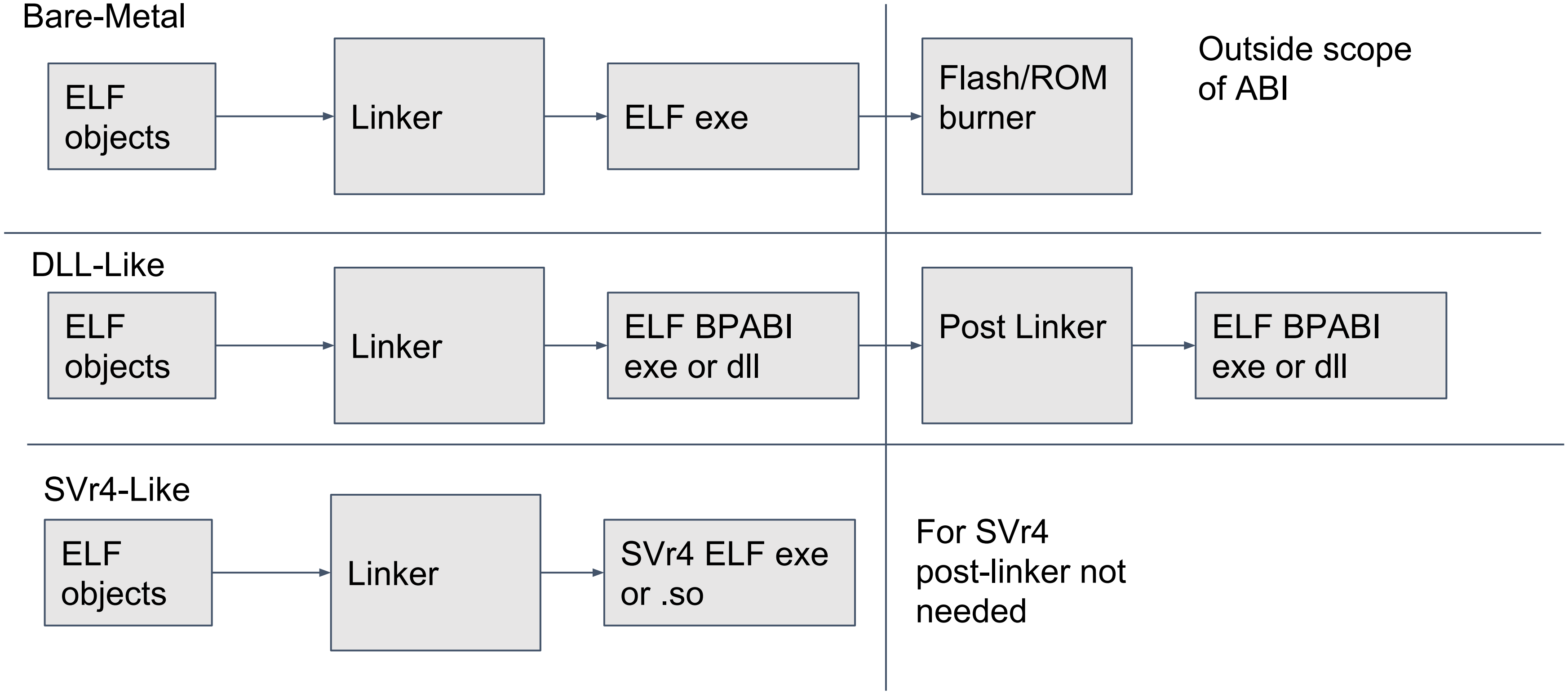


**Tag_ABI_enum_size**: int

**Tag_ABI_enum_size**: short

warning: t2.o uses variable-size enums yet the output is to use 32-bit enums; use of enum values across objects may fail

# Base Platform ABI for the ARM Architecture



Bare-Metal

ELF objects → Linker → ELF exe → Flash/ROM burner

Outside scope of ABI

DLL-Like

ELF objects → Linker → ELF BPABI exe or dll → Post Linker → ELF BPABI exe or dll

SVr4-Like

ELF objects → Linker → SVr4 ELF exe or .so

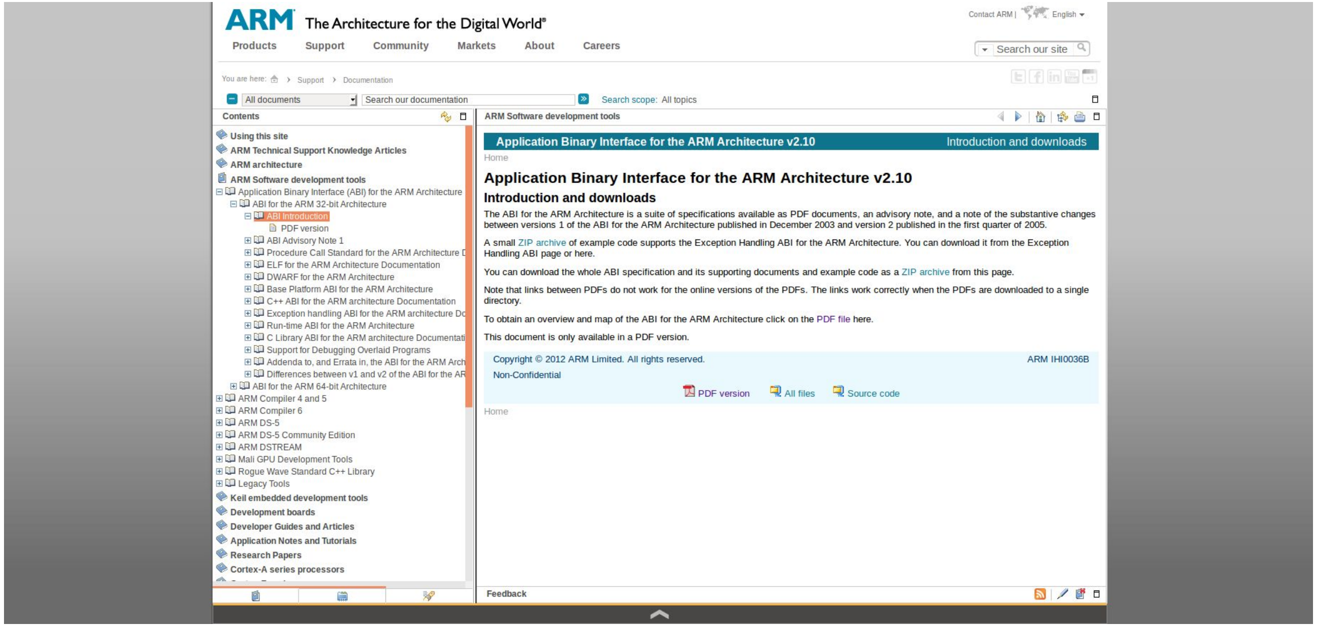
For SVr4 post-linker not needed

# Navigating the ARM ABI

- How to find the information you need in the set of documents making up the ABI?

# How to navigate the ABI in general

- The “Application Binary Interface for the ARM Architecture The Base Standard”
  - Referred to as Introduction in infocenter.arm.com
- ABI is available under ARM Software Development Tools



ARM The Architecture for the Digital World®

Contact ARM | English

Products Support Community Markets About Careers

Search our site

You are here: › Support › Documentation

All documents | Search our documentation | Search scope: All topics

Contents

- Using this site
- ARM Technical Support Knowledge Articles
- ARM architecture
- ARM Software development tools
  - Application Binary Interface (ABI) for the ARM Architecture
    - ABI for the ARM 32-bit Architecture
      - ABI Introduction
        - PDF version
      - ABI Advisory Note 1
      - Procedure Call Standard for the ARM Architecture D
      - ELF for the ARM Architecture Documentation
      - DWARF for the ARM Architecture
      - Base Platform ABI for the ARM Architecture
      - C++ ABI for the ARM architecture Documentation
      - Exception handling ABI for the ARM architecture Do
      - Run-time ABI for the ARM Architecture
      - C Library ABI for the ARM architecture Documentati
      - Support for Debugging Overlaid Programs
      - Addenda to, and Errata in, the ABI for the ARM Arch
      - Differences between v1 and v2 of the ABI for the AR
    - ABI for the ARM 64-bit Architecture
  - ARM Compiler 4 and 5
  - ARM Compiler 6
  - ARM DS-5
  - ARM DS-5 Community Edition
  - ARM DSTREAM
  - Mali GPU Development Tools
  - Rogue Wave Standard C++ Library
  - Legacy Tools
- Keil embedded development tools
- Development boards
- Developer Guides and Articles
- Application Notes and Tutorials
- Research Papers
- Cortex-A series processors

ARM Software development tools

Application Binary Interface for the ARM Architecture v2.10 | Introduction and downloads

Home

## Application Binary Interface for the ARM Architecture v2.10

### Introduction and downloads

The ABI for the ARM Architecture is a suite of specifications available as PDF documents, an advisory note, and a note of the substantive changes between versions 1 of the ABI for the ARM Architecture published in December 2003 and version 2 published in the first quarter of 2005.

A small ZIP archive of example code supports the Exception Handling ABI for the ARM Architecture. You can download it from the Exception Handling ABI page or here.

You can download the whole ABI specification and its supporting documents and example code as a ZIP archive from this page.

Note that links between PDFs do not work for the online versions of the PDFs. The links work correctly when the PDFs are downloaded to a single directory.

To obtain an overview and map of the ABI for the ARM Architecture click on the PDF file here.

This document is only available in a PDF version.

Copyright © 2012 ARM Limited. All rights reserved. ARM IHI0036B

Non-Confidential

PDF version All files Source code

Home

Feedback

# How to navigate the ABI as a programmer

- In an ideal world you won't have to, goal of the ABI is that most things should just work
- When things don't just work the ABI can be a good source of information to help track down the problem
    - Diagnosing compatibility problems between tools
    - Understanding error messages from low-level tools such as assemblers and linkers
- ELF for the ARM Architecture [**AAELF**] is usually the first port of call for linker and non-syntax assembler error messages
- Addenda to and errata in the ABI for the ARM Architecture [**ADDENDA**] contains the Build Attributes values that may explain link time compatibility messages.
- [**AAPCS**] For how to call a C/C++ function from Assembler

# Concluding thoughts

- Looking back at the 32-bit ABI
- Influences on the 64-bit ABI
- References

# Looking back on the 32-bit ABI

- Recall that back in the early 2000s the hardware and software landscape looked very different
- Large amount of consolidation has occurred at all levels
- Much more sharing done at the source code level
- The problem of the RTOS having to ship 20 different binaries for 20 different toolchains is much reduced

# The 64-bit ABI for the ARM Architecture

- A chance to start from scratch without baggage of existing implementations
  - **AAELF**, **AADWARF**, **AAPCS** and **CPPABI** minimum necessary building blocks for SW tools to aim for
- AArch64 only available in the A profile
  - Reusing existing industry standards with minimal changes acceptable.
- Primary use case is to run rich software platforms such as Linux and Android
  - Unit of sharing is either shared-libraries with C-like exported interfaces or source code
- Number of developers writing applications for a platform vastly outnumber OS and bare-metal developers
  - Platform specific standards and interfaces more important than defining a base platform ABI with post-linking

# Conclusion

- The 32-bit ABI for the ARM Architecture is an embedded ABI
  - Interoperability at the binary package level
- Platforms may build their own standards on top of the ABI
  - Tends to be the dominant toolchain on a particular platform rather than documentation
- The 64-bit ABI is more traditional, concentrating on what platforms need to build upon.

# References

- ARM published
  - http://infocenter.arm.com/help/topic/com.arm.doc.subset.swdev.abi/index.html
  - ABI for the ARM 32-bit Architecture
  - ABI for the ARM 64-bit Architecture
    - Release 1.0
      - Incorporating PCS, ELF, Dwarf and C++
    - Release 1.1 Beta
      - PCS and ELF changes for ILP32
  - ACLE http://infocenter.arm.com/help/topic/com.arm.doc.ihi0053d/index.html
- Generic Standards
  - ELF http://www.sco.com/developers/gabi/
  - Dwarf http://www.dwarfstd.org/
  - Itanium C++ ABI https://mentorembedded.github.io/cxx-abi/abi.html
  - Thread local storage https://www.akkadia.org/drepper/tls.pdf
  - Thread local storage descriptors https://www.fsfla.org/~lxoliva/writeups/TLS/RFC-TLSDESC-ARM.txt

# Thank You

#BUD17
For further information: www.linaro.org
BUD17 keynotes and videos on: connect.linaro.org