

Arm® Development Studio

Version 2019.0

Getting Started Guide



Arm® Development Studio

Getting Started Guide

Copyright © 2018, 2019 Arm Limited or its affiliates. All rights reserved.

Release Information

Document History

Issue	Date	Confidentiality	Change
1800-00	27 November 2018	Non-Confidential	First release for Arm® Development Studio
1800-01	18 December 2018	Non-Confidential	Documentation update 1 for Arm® Development Studio 2018.0
1800-02	31 January 2019	Non-Confidential	Documentation update 2 for Arm® Development Studio 2018.0
1900-00	11 April 2019	Non-Confidential	Updated document for Arm® Development Studio 2019.0

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, third party 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 © 2018, 2019 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

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

<http://www.arm.com>

Contents

Arm® Development Studio Getting Started Guide

Preface

<i>About this book</i>	9
------------------------------	---

Chapter 1

Introduction to Arm® Development Studio

1.1	<i>Arm® Compiler</i>	1-13
1.2	<i>Arm® Debugger</i>	1-14
1.3	<i>Debug probes</i>	1-15
1.4	<i>Fixed Virtual Platform models</i>	1-17
1.5	<i>Arm Streamline Performance Analyzer</i>	1-18
1.6	<i>Mali™ Graphics Debugger</i>	1-19

Chapter 2

Installation

2.1	<i>Hardware and host platform requirements</i>	2-21
2.2	<i>Debug system requirements</i>	2-22
2.3	<i>Installing on Windows</i>	2-23
2.4	<i>Installing on Linux</i>	2-25
2.5	<i>Additional Linux libraries</i>	2-26

Chapter 3

Licensing Arm® Development Studio

3.1	<i>Using Product Setup to add a license</i>	3-28
3.2	<i>Viewing and managing licenses</i>	3-29

Chapter 4

Introduction to the Integrated Development Environment

4.1	<i>Integrated Development Environment (IDE) Overview</i>	4-32
-----	--	------

	4.2	Using the IDE	4-33
	4.3	Language settings	4-37
Chapter 5		Tutorials	
	5.1	Tutorial: Hello World	5-39
Appendix A		Terminology and Shortcuts	
	A.1	Terminology	Appx-A-54
	A.2	Keyboard shortcuts	Appx-A-56

List of Figures

Arm® Development Studio Getting Started Guide

Figure 3-1	Adding a license in preferences dialog box.	3-29
Figure 3-2	Deleting a license in preferences dialog box.	3-30
Figure 4-1	IDE in the Development Studio perspective.	4-32
Figure 4-2	Workspace Launcher dialog box	4-33
Figure 4-3	Open Perspective dialog box	4-34
Figure 4-4	Adding a view in an area	4-35
Figure 4-5	Adding a view in [armds]	4-35
Figure 5-1	Screenshot of the IDE after creating a new project	5-41
Figure 5-2	Select VE_Cortex_A9x1 model	5-43
Figure 5-3	Edit configuration Connection tab	5-44
Figure 5-4	Edit configuration Files tab	5-45
Figure 5-5	Select helloworld.axf file	5-46
Figure 5-6	Debug from symbol main	5-47
Figure 5-7	Debug Control View	5-47
Figure 5-8	main () in code editor	5-48
Figure 5-9	Target console output	5-49
Figure 5-10	Commands view	5-49
Figure 5-11	Code Editor view	5-50
Figure 5-12	Disassembly view	5-50
Figure 5-13	Memory view	5-51
Figure 5-14	Disconnecting from a target using the Debug Control view	5-51
Figure 5-15	Disconnecting from a target using the Commands view	5-52

List of Tables

Arm® Development Studio Getting Started Guide

Table 2-1 *Linux kernel version requirements* 2-22

Preface

This preface introduces the *Arm® Development Studio Getting Started Guide*.

It contains the following:

- [About this book on page 9](#).

About this book

This book describes how to get started with Arm® Development Studio. It takes you through the processes of installing and licensing Arm® Development Studio, and guides you through some of the common tasks that you might encounter when using Arm® Development Studio for the first time.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction to Arm® Development Studio

Arm Development Studio is a professional software development solution for bare-metal embedded systems and Linux-based systems. It covers all stages in development from boot code and kernel porting to application and bare-metal debugging, including performance analysis.

Chapter 2 Installation

Arm Development Studio is available for Windows and Linux operating systems. This chapter covers installation requirements and the installation process.

Chapter 3 Licensing Arm® Development Studio

Arm Development Studio uses the FlexNet license management software to enable features that correspond to specific editions.

Chapter 4 Introduction to the Integrated Development Environment

The Arm Development Studio Integrated Development Environment (IDE) is Eclipse-based, combining the Eclipse IDE from the Eclipse Foundation with the compilation and debug technology of Arm tools.

Chapter 5 Tutorials

Contains tutorials to help you get started with Arm Development Studio.

Appendix A Terminology and Shortcuts

Supplementary information for new users of Arm Development Studio.

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the [Arm® Glossary](#) for more information.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

monospace

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

monospace bold

Denotes language keywords when used outside example code.

<and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments.
For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title *Arm Development Studio Getting Started Guide*.
- The number 101469_1900_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

———— **Note** ————

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information

- [Arm® Developer](#).
- [Arm® Information Center](#).
- [Arm® Technical Support Knowledge Articles](#).
- [Technical Support](#).
- [Arm® Glossary](#).

Chapter 1

Introduction to Arm® Development Studio

Arm Development Studio is a professional software development solution for bare-metal embedded systems and Linux-based systems. It covers all stages in development from boot code and kernel porting to application and bare-metal debugging, including performance analysis.

It includes:

The Arm Compiler 5 and Arm Compiler 6 toolchains.

These enable you to build embedded and bare-metal embedded applications.

Arm Debugger.

A graphical debugger supporting software development on Arm processor-based targets and Fixed Virtual Platform (FVP) targets.

Fixed Virtual Platform (FVP) targets.

Single and multi-core simulation models for architectures Armv6-M, Armv7-A/R/M and Armv8-A/R/M. These enable you to develop software without any hardware.

Streamline.

A graphical performance analysis tool that enables you to transform sampling data and system trace into reports that present data in both visual and statistical forms.

Mali™ Graphics Debugger.

The Mali Graphics Debugger allows graphics developers to trace OpenGL ES, Vulkan and OpenCL API calls in their applications.

Dedicated examples, applications, and supporting documentation to help you get started with using Arm Development Studio tools.

Some third-party compilers are compatible with Arm Development Studio. For example, the GNU Compiler tools enable you to compile bare-metal, Linux kernel, and Linux applications for Arm targets.

It contains the following sections:

- [1.1 Arm® Compiler on page 1-13.](#)
- [1.2 Arm® Debugger on page 1-14.](#)
- [1.3 Debug probes on page 1-15.](#)
- [1.4 Fixed Virtual Platform models on page 1-17.](#)
- [1.5 Arm Streamline Performance Analyzer on page 1-18.](#)
- [1.6 Mali™ Graphics Debugger on page 1-19.](#)

1.1 Arm® Compiler

Arm Compiler tools enable you to build applications and libraries suitable for bare-metal embedded systems.

Arm Development Studio provides two versions of Arm Compiler for compiling embedded and bare-metal embedded applications:

- Arm Compiler 5 - Supports all Arm architectures from Armv4 to Armv7 inclusive.

————— **Note** —————

All architectures before Armv4 are obsolete and are no longer supported by Arm Compiler 5.

- Arm Compiler 6 - Supports Armv6-M, Armv7, and Armv8 architectures. Arm Compiler 6 also supports the *SVE architectural extensions*. This toolchain is recommended for software development targeting Armv8-A with the SVE architectural extensions for High Performance Computing (HPC).

You can run the compilers within the Arm Development Studio IDE, or from the command line.

————— **Note** —————

The features available to you in Arm Compiler depend on your individual license type.

For example, a license might:

- Limit the use of Arm Compiler to specific processors.
- Place a maximum limit on the size of images that can be produced.

You can enable additional features of Arm Compiler by purchasing a license for the full Arm Development Studio suite. Contact your tools supplier for details.

Related information

Add a compiler to Arm Development Studio

1.2 Arm® Debugger

Arm Debugger is accessible using either the Arm Development Studio IDE or command-line, and supports software development on Arm processor-based targets and Fixed Virtual Platform (FVP) targets.

Using Arm Debugger through the IDE allows you to debug bare-metal and Linux applications with comprehensive and intuitive views, including:

- Synchronized source and disassembly.
- Call stack.
- Memory.
- Registers.
- Expressions.
- Variables.
- Threads.
- Breakpoints.
- Trace.

The **Debug Control** view enables you to single-step through applications at source-level or instruction-level, and see other views update when the code is executed. Setting breakpoints or watchpoints stops the application and allows you to explore the behaviour of the application. You can also use the view to trace function executions in your application with a timeline showing the sequence of events, if supported by the target.

You can also debug using the **Arm DS Command Prompt** command-line console, which allows for automation of debug and trace activities through scripts.

Related information

Debug control view

Overview of Arm Debugger

1.3 Debug probes

Arm Development Studio supports various debug adapters and connections.

Debug adapters

Debug adapters vary in complexity and capability. When you use them with Arm Development Studio, they provide high-level debug functionality, for example:

- Reading/writing registers.
- Setting breakpoints.
- Reading from memory.
- Writing to memory.

Supported Arm debug adapters include:

- Arm DSTREAM.
- Arm DSTREAM-ST.
- Keil® ULINK™ 2.
- Keil® ULINK pro.
- Keil® ULINK pro D.

Debug connections

Debug connections allow the debugger to debug a variety of targets.

Supported debug connections include:

- CADI (debug interface for models).
- Ethernet to gdbserver.
- CMSIS-DAP.
- DTS adviceLUNA (JTAG ICE).
- Altera USB-Blaster II.

Note

Arm Debugger can connect to Altera Arria V SoC, Arria 10 SoC, Cyclone V SoC and Stratix 10 boards using Altera USB-Blaster and USB-Blaster II debug units.

To enable the connections, ensure that the environment variable `QUARTUS_ROOTDIR` is set and contains the path to the Altera Quartus tools installation directory:

- On Windows, this environment variable is usually set by the Quartus tools installer.
- On Linux, you might have to manually set the environment variable to the Altera Quartus tools installation path. For example, `~/altera/13.0/qprogrammer`.

For information on installing device drivers for USB-Blaster and USB-Blaster II, consult your Altera Quartus tools documentation.

Debug hardware configuration

Use the debug hardware configuration views in Arm Development Studio to update and configure the debug hardware adapter that provides the interface between your development target and your workstation.

Arm Development Studio provides the following views:

- **Debug Hardware Config IP view**

Use this view to *configure the IP address* on a debug hardware adapter.

- **Debug Hardware Firmware Installer view**

Use this view to *update the firmware* on a debug hardware adapter.

Note

These views only support the DSTREAM family of devices.

1.4 Fixed Virtual Platform models

Fixed Virtual Platforms (FVPs) are complete simulations of an Arm system, including processor, memory and peripherals. FVP targets give you a comprehensive model on which to build and test your software, from the view of a programmer.

When using an FVP, absolute timing accuracy is sacrificed to achieve fast simulated execution speed. This means that you can use a model for confirming software functionality, but you must not rely on the accuracy of cycle counts, low-level component interactions, or other hardware-specific behavior.

Arm Development Studio provides several FVPs, covering a range of processors in the Cortex® family. You can also connect to a variety of other Arm and third-party simulation models that implement CADI.

Arm Development Studio includes an Armv8-A FVP executable that supports the SVE architecture extension. The executables are located in <install_directory>\bin\... You can use them to run your applications from either the command line or within the Arm Development Studio IDE.

Related information

About the Component Architecture Debug Interface (CADI)

1.5 Arm Streamline Performance Analyzer

Arm Streamline Performance Analyzer is a graphical performance analysis tool. It enables you to transform sampling data, instruction trace, and system trace into reports that present the data in both visual and statistical forms.

Streamline uses hardware performance counters with kernel metrics to provide an accurate representation of system resources.

1.6 Mali™ Graphics Debugger

Mali Graphics Debugger is a tool to help OpenGL® ES, EGL™, OpenCL™, and Vulkan™ developers get the best out of their applications through analysis at the API level.

Mali Graphics Debugger allows developers to trace OpenGL ES, Vulkan and OpenCL API calls in their application and understand frame-by-frame the effect on the application to help identify possible issues. Attempted misuse of the API is highlighted, as are recommendations for improvement on a Mali-based system. Trace information may also be captured to a file on one system and be analyzed later. The state of the underlying GPU subsystem is observable at any point.

Chapter 2

Installation

Arm Development Studio is available for Windows and Linux operating systems. This chapter covers installation requirements and the installation process.

It contains the following sections:

- [2.1 Hardware and host platform requirements](#) on page 2-21.
- [2.2 Debug system requirements](#) on page 2-22.
- [2.3 Installing on Windows](#) on page 2-23.
- [2.4 Installing on Linux](#) on page 2-25.
- [2.5 Additional Linux libraries](#) on page 2-26.

2.1 Hardware and host platform requirements

For the best experience with Arm Development Studio, your hardware and host platform should meet the minimum requirements.

Hardware requirements

To install and use Arm Development Studio, your workstation must have at least:

- A dual core x86 2GHz processor (or equivalent).
- 2GB of RAM.
- Approximately 3GB of hard disk space.

To improve performance, Arm recommends a minimum of 4GB of RAM when you :

- Debug large images.
- Use models with large simulated memory maps.
- Use Streamline.

Host platform requirements

Arm Development Studio supports the following host platforms:

- Windows 7 SP1 Professional Edition
- Windows 7 SP1 Enterprise Edition
- Windows 10
- Red Hat Enterprise Linux 6 Workstation
- Red Hat Enterprise Linux 7 Workstation
- Ubuntu Desktop Edition 16.04 LTS
- Ubuntu Desktop Edition 18.04 LTS

Note

Arm Development Studio only supports 64-bit host platforms.

Arm® Compiler host platform requirements

Arm Development Studio contains the latest versions of Arm Compiler 5 and Arm Compiler 6. The release notes for each compiler version provide information on host platform compatibility:

- [Arm Compiler 5](#)
- [Arm Compiler 6](#)

For information on adding other versions of Arm Compiler 5 and Arm Compiler 6 to Arm Development Studio, see [register a compiler toolchain](#).

2.2 Debug system requirements

When debugging bare-metal and Linux targets, you need additional software and hardware.

Bare-metal requirements

You require a debug unit to connect bare-metal targets to Arm Development Studio.

Arm Development Studio supports the following debug units:

- DSTREAM
- DSTREAM-ST
- ULINK pro
- ULINK pro D
- ULINK 2

Linux application and Linux kernel requirements

Linux application debug requires gdbserver version 7.0 or later on your target.

In addition to gdbserver, certain architecture and debug features have minimum Linux kernel version requirements. This is shown in the following table:

Table 2-1 Linux kernel version requirements

Architecture or debug feature	Minimum Arm Linux kernel version
Debug with Arm Debugger	2.6.28
Application debug on Symmetric MultiProcessing (SMP) systems	2.6.36
Access VFP and Arm Neon™ registers	2.6.30
Streamline	3.4

Managing firmware updates

- For DSTREAM, use the [debug hardware firmware installer view](#) to check the firmware and update it if necessary. Updated firmware is available in <install_directory>/sw/debughw/firmware.
- To use ULINK 2 debug probe with Arm Debugger, you must upgrade with CMSIS-DAP compatible firmware. On Windows, the UL2_Upgrade.exe program can upgrade your ULINK 2 unit. The program and instructions are available in <install_directory>/sw/debughw/ULINK2.
- For ULINK pro and ULINK pro D, Arm Development Studio manages the firmware installation.

2.3 Installing on Windows

There are two ways to install Arm Development Studio, you can use either the installation wizard, or the command line.

Note

You can install multiple versions of Arm Development Studio on Windows platforms. To do this, you must use different root installation directories.

This section contains the following subsections:

- [2.3.1 Using the installation wizard on page 2-23.](#)
- [2.3.2 Using the command line on page 2-23.](#)

2.3.1 Using the installation wizard

To install Arm Development Studio on Windows using the installation wizard, use the following procedure.

Prerequisites

- [Download](#) the Arm Development Studio installation package.

Procedure

1. Unzip the downloaded .zip file.
2. Run **armds-<version>.exe** from this location.
3. Follow the on-screen instructions.

Note

- During installation, you might be prompted to install device drivers. Arm recommends that you install these drivers. They allow USB connections to DSTREAM and Energy Probe hardware units. They also support networking for the simulation models. These drivers are required to use these features.
 - When the drivers are installed, you might see some warnings about driver software. You can safely ignore these warnings.
-

2.3.2 Using the command line

To install Arm Development Studio on Windows using the command line, use the following procedure.

Prerequisites

- [Download](#) the Arm Development Studio installation package.
- You must have admin privileges on your machine to install from the command line.

Procedure

1. Open the command prompt, with administrative privileges.
2. Run the Microsoft installer, **msiexec.exe**.

Note

- You must provide the location of the .msi file as an argument to **msiexec**.
 - To display a full list of **msiexec** options, run **msiexec /?** from the command line.
-

Example 2-1 Examples

An example of how to install Arm Development Studio using **msiexec** is:

```
msiexec.exe /i <installer_locationdatainstall.msi> EULA=1 /qn /l*v install.log
```

Command	Definition
/i	Performs the installation.
<installer_locationdatainstall.msi>	Specifies the full path name of the .msi file to install.
/EULA=1	This is an Arm-specific option. Set EULA to 1 to accept the End User License Agreement (EULA). You must read the EULA before accepting it on the command line. This can be found in the GUI installer, the installation files, or on the Arm Development Studio downloads page.
/qn	Specifies quiet mode; installation does not require user interaction. ————— Note ————— Device driver installation requires user interaction. If you do not require USB drivers, or if you want the installation to avoid user interaction for USB drivers, use the SKIP_DRIVERS=1 option on the command line. —————
/l*v<install.log>	Specifies the log file to display all outputs from the installation.

2.4 Installing on Linux

You can install one or more versions of Arm Development Studio on Linux.

To install Arm Development Studio on Linux, use the following procedure after [downloading](#) the installation package.

Note

You can install multiple versions of Arm Development Studio on Linux platforms. To do this, you must use different root installation directories.

Procedure

1. Navigate to `<install_location>/sw/dependency_check/dependency_check_linux-x86_64.sh`. Run the `dependency_check_linux-x86_64.sh` script. This script prints a list of any missing dependencies that must be installed before continuing.

Note

Tools installed by the 64-bit installer have dependencies on 32-bit system libraries. Arm Development Studio tools might fail to run, or might report errors about missing libraries if 32-bit compatibility libraries are not installed. You can find a full list of libraries in [Additional Linux libraries on page 2-26](#).

2. Run `armds-<version>.sh`. Follow the on-screen instructions.

Next Steps

Device drivers and desktop shortcuts are optional features. The device drivers allow USB connections to debug hardware units, for example DSTREAM. The desktop menu is created using the <http://www.freedesktop.org/> menu system on supported Linux platforms.

To install optional features post-install by using root privileges, run:

```
run_post_install_for_Arm_DS_IDE_<version>.sh
```

This script is found in the install directory.

Note

Use `suite_exec` to configure the environment variables correctly for Arm Development Studio. For example, run `<install_directory>/bin/suite_exec <shell>` to open a shell with the PATH and other environment variables correctly configured. Run `suite_exec` with no arguments for more help.

2.5 Additional Linux libraries

To install Arm Development Studio on Linux, you need to install some additional libraries, which might not be installed on your system.

The specific libraries that require installation depend on the distribution of Linux that you are running. The `dependency_check_linux-x86_64.sh` script identifies libraries you must install. For more information, refer to [Installing on Linux on page 2-25](#).

Note

If the required libraries are not installed, some of the Arm Development Studio tools might fail to run. You might encounter error messages, such as:

- `armcc: No such file or directory`
 - `arm-linux-gnueabi-hf-gcc: error while loading shared libraries: libstdc++.so.6: cannot open shared object file: No such file or directory`
-

Required libraries

Arm Development Studio depends on the following libraries:

- `libasound.so.2`
- `libatk-1.0.so.0`
- `libc.so.6 *`
- `libcairo.so.2`
- `libfontconfig.so.1`
- `libfreetype.so.6`
- `libgcc_s.so.1 *`
- `libGL.so.1`
- `libGLU.so.1`
- `libgthread-2.0.so.0`
- `libgtk-x11-2.0.so.0`
- `libncurses.so.5`
- `libnsl.so.1`
- `libstdc++.so.6 *`
- `libusb-0.1.so.4`
- `libX11.so.6`
- `libXext.so.6`
- `libXi.so.6`
- `libXrender.so.1`
- `libXt.so.6`
- `libXtst.so.6`
- `libz.so.1 *`

Note

On a 64-bit installation, libraries marked with an asterisk require an additional 32-bit compatibility library.

Some components also render using a browser library. Arm recommends that you install one of these libraries to ensure all components render correctly:

- `libwebkit-1.0.so.2`
- `libwebkitgtk-1.0.so.0`
- `libxpcos.so`

Chapter 3

Licensing Arm® Development Studio

Arm Development Studio uses the FlexNet license management software to enable features that correspond to specific editions.

To compare Arm Development Studio editions, see [Compare editions](#)

It contains the following sections:

- [3.1 Using Product Setup to add a license](#) on page 3-28.
- [3.2 Viewing and managing licenses](#) on page 3-29.

3.1 Using Product Setup to add a license

When you first open Arm Development Studio, the **Product Setup** dialog box opens and prompts you to add a license.

Prerequisites

If you have purchased Arm Development Studio, you need the license server address and port number, or the license file.

Procedure

1. Select **Add product license** and click **Next**.
2. Add your license:
 - For a license server, select **License server** and enter the license server address and port number, in the form <port number> @ <server address>. Click **Next**.
 - For a license file, select **License file** and then **Browse...** to select the file from your workstation. Click **Next**.
 - For an evaluation license:
 1. Select **Obtain evaluation license** and click **Next**.
 2. Log into your Arm account and click **Next**.
 3. Choose a network interface and click **Finish**. An evaluation license is generated.
3. Select a product to activate, and click **Finish**.

3.2 Viewing and managing licenses

To view license information within Arm Development Studio, select **Help > Arm License Manager**.

This section contains the following subsections:

- [3.2.1 Adding a license on page 3-29.](#)
- [3.2.2 Deleting a license on page 3-29.](#)

3.2.1 Adding a license

You can add a license to Arm Development Studio using the Arm License Manager.

Prerequisites

You need the license server address and port number, or the license file.

Procedure

1. Click **Help > Arm License Manager** to view your license information.
2. Click **Add** to open the **Product Setup** wizard.

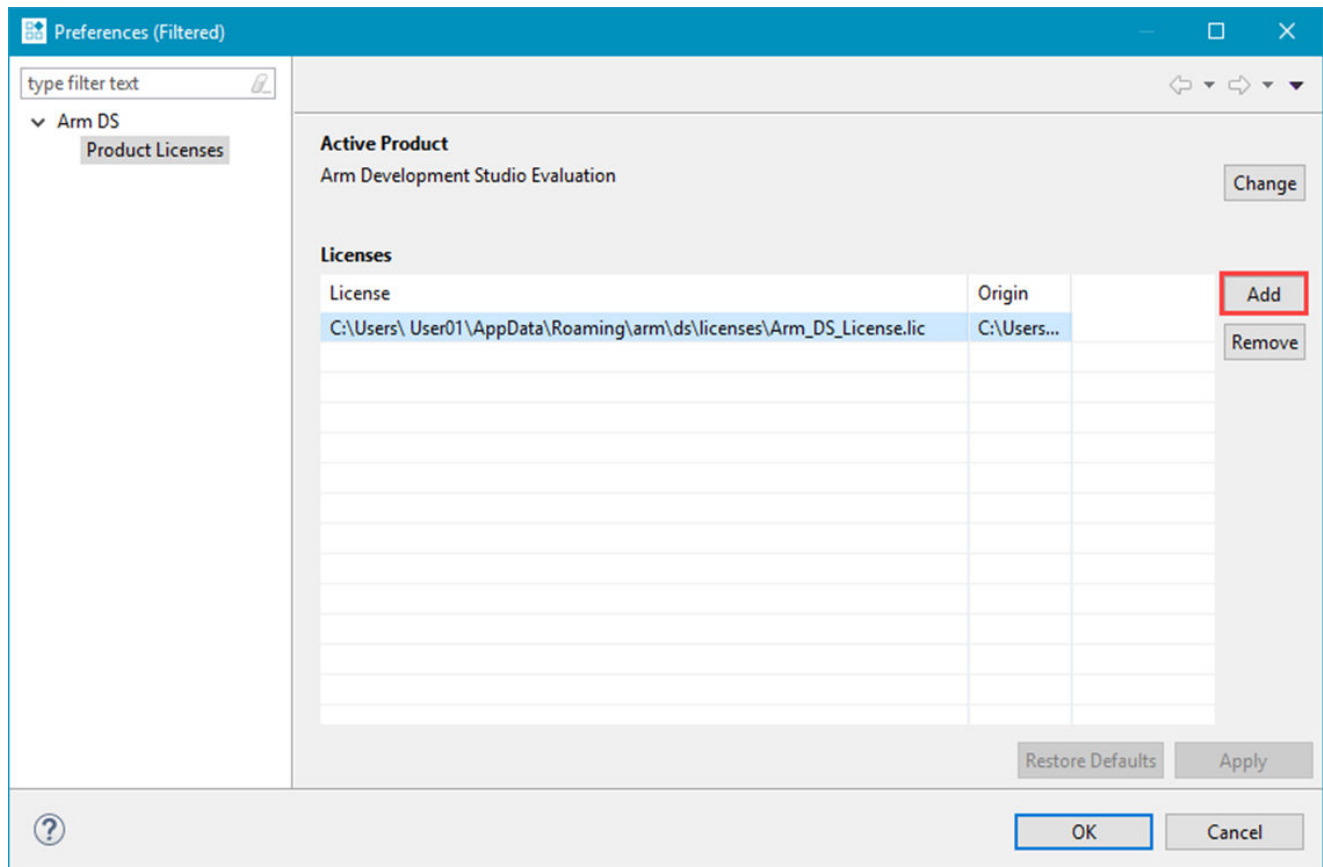


Figure 3-1 Adding a license in preferences dialog box.

3. Follow the steps in [Using Product Setup to add a license on page 3-28](#) to add your license.

Related tasks

- 3.1 [Using Product Setup to add a license on page 3-28](#)

3.2.2 Deleting a license

You can use the Arm license manager to delete unwanted licenses from Arm Development Studio.

Procedure

1. Click **Help > Arm License Manager** to view your license information.
2. Select the license you want to delete, and click **Remove**.

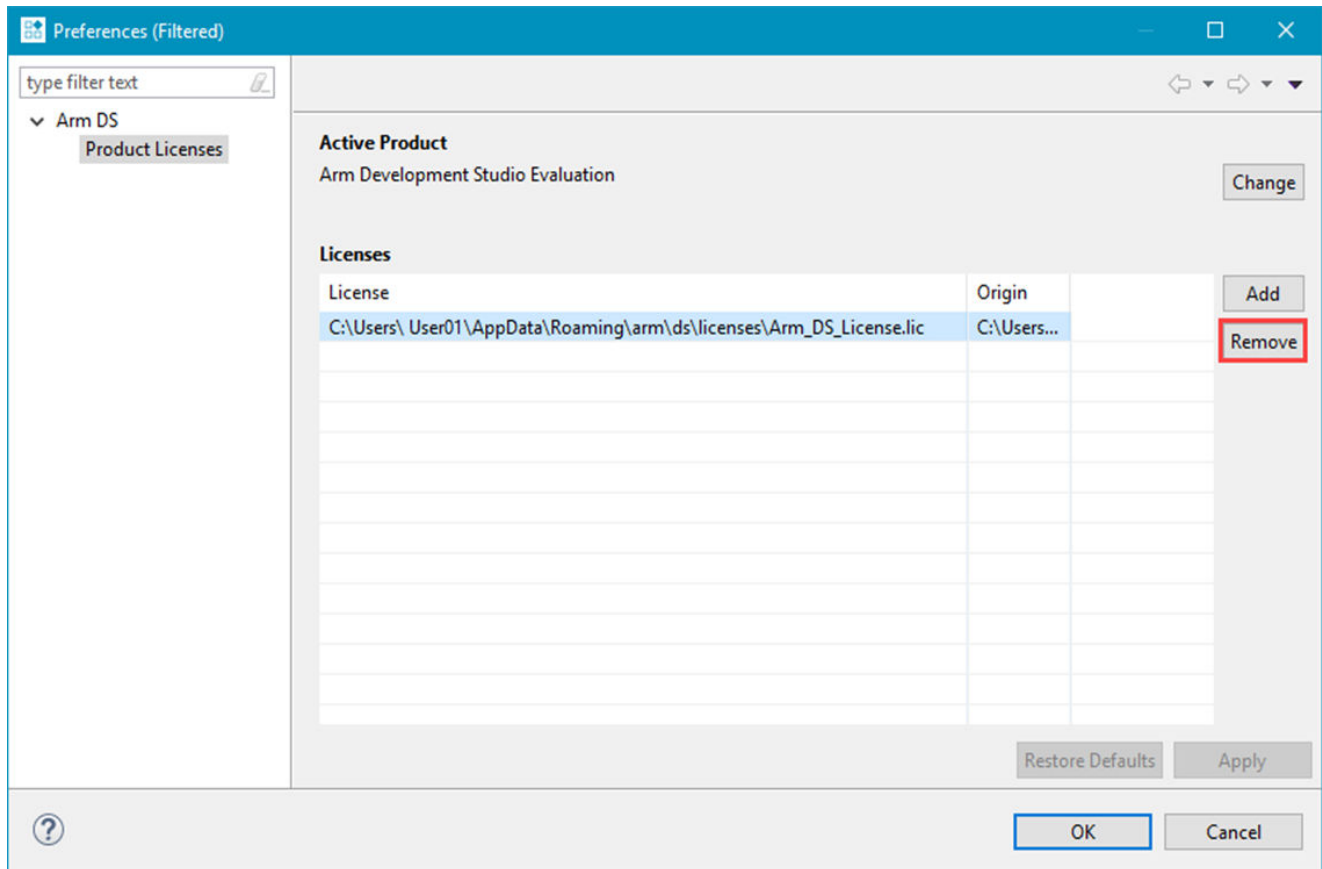


Figure 3-2 Deleting a license in preferences dialog box.

Chapter 4

Introduction to the Integrated Development Environment

The Arm Development Studio Integrated Development Environment (IDE) is Eclipse-based, combining the Eclipse IDE from the Eclipse Foundation with the compilation and debug technology of Arm tools.

It includes:

Project Explorer

The project explorer enables you to perform various project tasks such as adding or removing files and dependencies to projects, importing, exporting, or creating projects, and managing build options.

Editors

Editors enable you to read, write, or modify C/C++ or Arm assembly language source files.

Perspectives and views

Perspectives provide customized views, menus, and toolbars to suit a particular type of environment. Arm Development Studio uses the **Development Studio** perspective by default. To switch perspectives, from the main menu, select **Window > Open Perspective**.

It contains the following sections:

- [4.1 Integrated Development Environment \(IDE\) Overview on page 4-32.](#)
- [4.2 Using the IDE on page 4-33.](#)
- [4.3 Language settings on page 4-37.](#)

4.1 Integrated Development Environment (IDE) Overview

The IDE contains a collection of views that are associated with a specific perspective.

Arm Development Studio uses the **Development Studio** perspective as default.

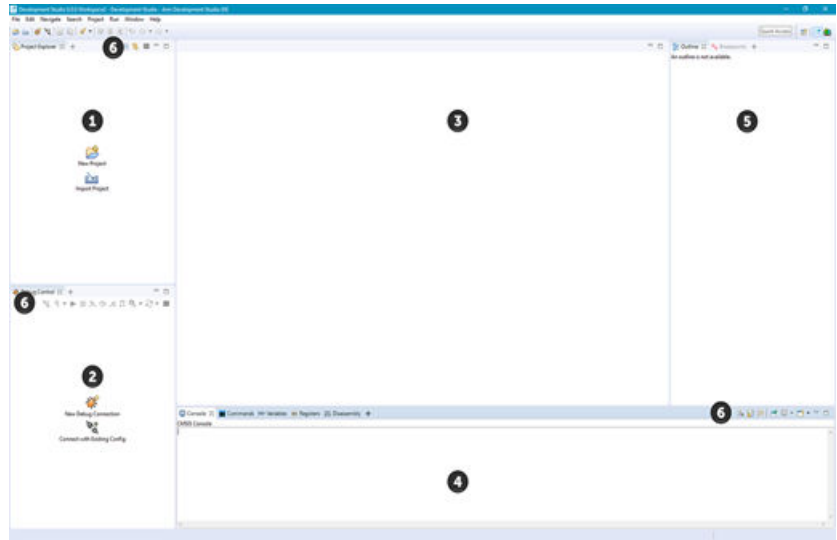


Figure 4-1 IDE in the Development Studio perspective.

1. Project Explorer - Use this view to create, build, and manage your projects.
2. Debug Control - Use this view to create and control debug connections.
3. Editor window - Use this view to view and modify the content of your source code files. The tabs in the editor area show files that are currently open for editing.
4. Debug views - Views specific to Arm Debugger. Multiple views can be stacked in an area, creating tabs.
5. Area containing additional debug views. All areas are fully customizable to contain any desired view.
6. Menus and Toolbars - The main menu and toolbar are located at the top of the IDE window. Other toolbars associated with specific features are located at the top of each perspective or view.

On exit, your settings save automatically. When you next open Arm Development Studio, the window returns to the same perspective and views.

For further information on a view, click inside it and press F1 to open the **Help** view.

Customize the IDE

You can customize the IDE by changing the layout, key bindings, file associations, and color schemes. These settings can be found in **Window > Preferences**. Changes are saved in your workspace. If you select a different workspace, then these settings might be different.

4.2 Using the IDE

The Arm Development Studio IDE can be customized. It is possible to choose the views you can see by following the instructions in this section.

This section contains the following subsections:

- [4.2.1 Changing the default workspace on page 4-33.](#)
- [4.2.2 Switching perspectives on page 4-33.](#)
- [4.2.3 Adding views on page 4-34.](#)

4.2.1 Changing the default workspace

The workspace is an area on your file system to store files and folders related to your projects, and your IDE settings. When Arm Development Studio launches for the first time, a default workspace is automatically created for you in `C:\Users\<user>\Development Studio Workspace`.

Note

Arm recommends that you select a dedicated workspace folder for your projects. If you select an existing folder containing resources that are not related to your projects, you cannot access them in Arm Development Studio. These resources might also cause a conflict later when you create and build projects.

Arm Development Studio automatically opens in the last used workspace.

Procedure

1. Select **File > Switch Workspace > Other...**. The **Eclipse Launcher** dialog box opens.

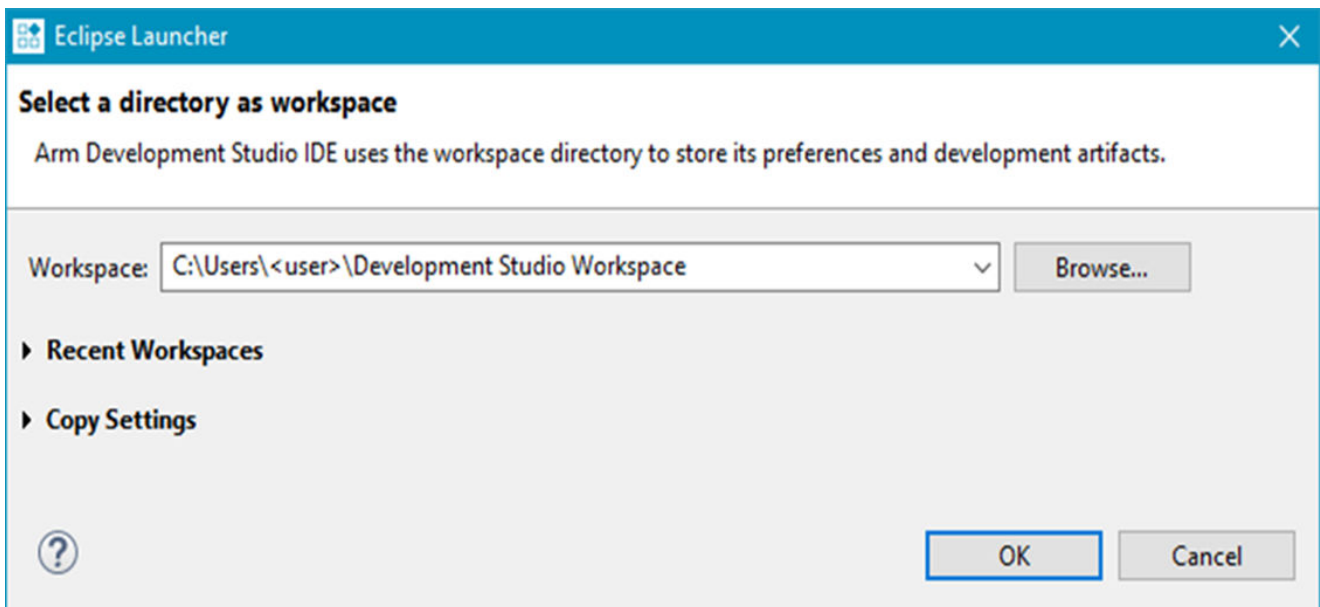


Figure 4-2 Workspace Launcher dialog box

2. Click **Browse...** to choose your workspace, and click **OK**.

Arm Development Studio relaunches in the new workspace.

4.2.2 Switching perspectives

Perspectives define the layout of your selected views and editors in the Arm Development Studio IDE. Each perspective has its own associated menus and toolbars.

Procedure

1. Go to **Window > Perspective > Open Perspective > Other...** This opens the **Open Perspective** dialog box.
2. Select the perspective that you want to open, and click **OK**.

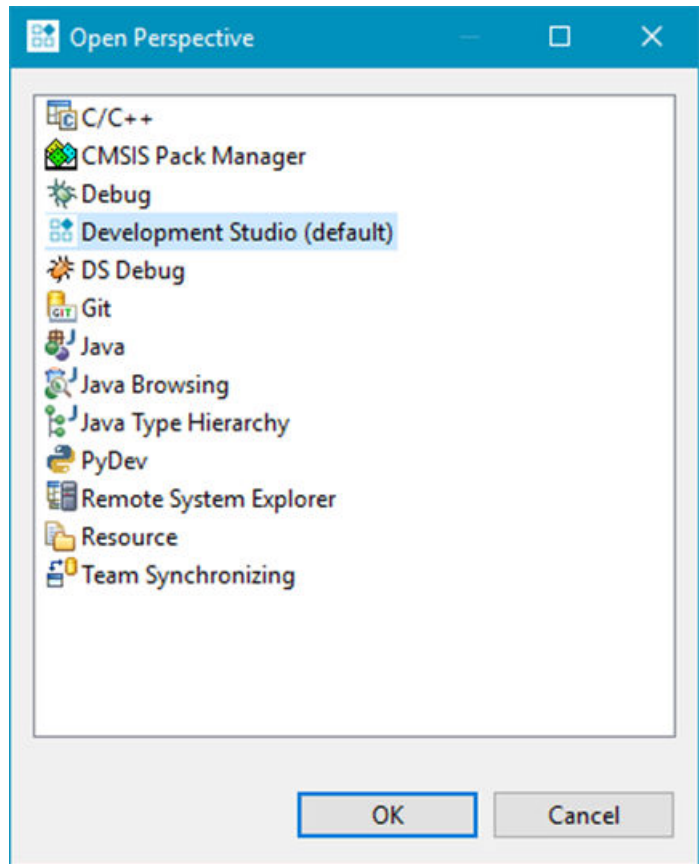


Figure 4-3 Open Perspective dialog box

Your perspective opens in the workspace.

Related information

Arm Debugger perspectives and views

4.2.3 Adding views

Views provide information for a specific function, corresponding to the active debug connection. Each perspective has a set of default views. You can add, remove, or reposition the views to customize your workspace.

Procedure

1. Click the + button in the area you want to add a view.

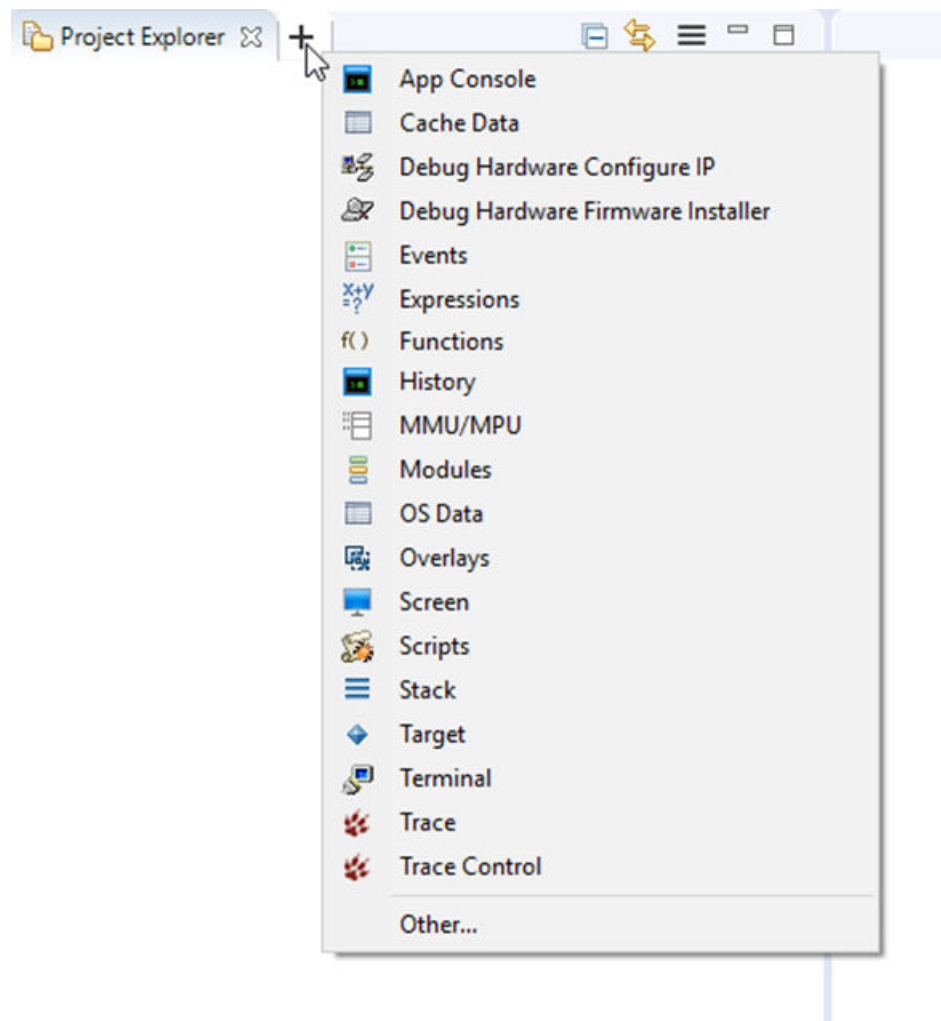


Figure 4-4 Adding a view in an area

2. Choose a view to add, or click **Other...** to open the **Show View** dialog box to see a complete list of available views.

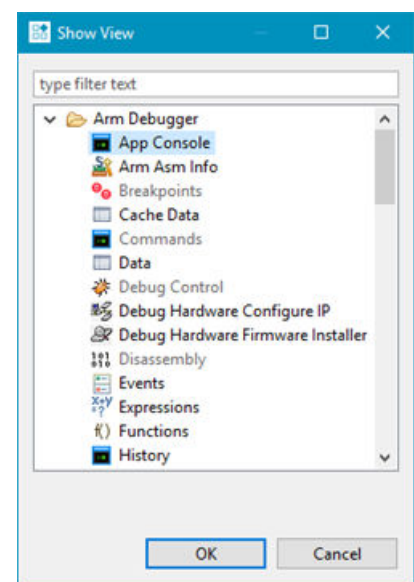


Figure 4-5 Adding a view in [armds]

3. Select the view you want to open, and click **OK**.

The view opens in the selected area.

Related information

Arm Debugger perspectives and views

4.3 Language settings

Some features of the IDE are translated into different languages. You can download language packs from the Eclipse website.

There are two ways to launch the IDE with a different language pack:

- If your operating system is running in the language that you want to use, then the IDE automatically displays the translated features.
- If your operating system is not running in the language that you want to use, then you must specify the `-nl` command-line argument when launching the IDE.

For example, to use the Japanese language pack, enter:

```
armds_ide -nl ja
```

Note

Arm Compiler 6 does not support Japanese characters in source files.

Chapter 5

Tutorials

Contains tutorials to help you get started with Arm Development Studio.

It contains the following section:

- [5.1 Tutorial: Hello World on page 5-39.](#)

5.1 Tutorial: Hello World

The Hello World tutorial is for new users, taking them through each step in getting their first project up and running.

This section contains the following subsections:

- [5.1.1 Open Arm® Development Studio for the first time on page 5-39.](#)
- [5.1.2 Create a project in C/C++ on page 5-40.](#)
- [5.1.3 Configure your project on page 5-41.](#)
- [5.1.4 Build your project on page 5-42.](#)
- [5.1.5 Configure your debug session on page 5-42.](#)
- [5.1.6 Application debug with Arm Debugger on page 5-48.](#)
- [5.1.7 Disconnecting from a target on page 5-51.](#)

5.1.1 Open Arm® Development Studio for the first time

The first time you open Arm Development Studio, you are prompted to add your license details. When you have completed the tasks in this section, you are ready to use Arm Debugger.

Arm Development Studio is available for both [Linux and Windows platforms on page 2-21](#).

Prerequisites

- Download and install Arm Development Studio, for either:
 - Linux: [Installing on Linux on page 2-25](#)
 - Windows: [Installing on Windows on page 2-23](#)
- If you have purchased Arm Development Studio, you need either your license file, or the address and port number of the license server you would like to connect to.

Procedure

1. Open Arm Development Studio:
 - On Windows, select **Windows menu > Arm Development Studio <version>**
 - On Linux:
 - GUI: Use your Linux variant's menu system to locate Arm Development Studio.
 - Command line: Run `<installation_directory>/bin/armds_ide`
2. The first time you open Arm Development Studio, the **Product Setup** dialog opens, which prompts you to add your product license. You can either:
 - **Add product license** - select this option if you have purchased Arm Development Studio.
 - **Obtain evaluation license** - select this option if you would like to evaluate the product.
3. Click **Next**.
4. If you selected **Add product license**:
 - a. Enter the location of your license file, or the address and port number of your license server, and click **Next**.
 - b. The Arm Development Studio editions that you are entitled to use are listed. Select the edition that you require, and click **Next**.
 - c. Check the details on the summary page. If they are correct, click **Finish**.
5. If you selected **Obtain evaluation license**:
 - a. Log into your Developer account using your Arm Developer account email address and password. If you do not have an account, click **Create an account**.
 - b. Select a network interface to which your license will be locked.
 - c. Click **Finish**.

Arm Development Studio opens. See [Integrated Development Environment \(IDE\) Overview on page 4-32](#), which describes the main features of the user interface.

Note

The workspace is automatically set by default, to either:

- Windows: <userhome>\Development Studio Workspace
- Linux: <userhome>/developmentstudio-workspace

You can change the default location by selecting **File > Switch Workspace**.

5.1.2 Create a project in C/C++

After installing and licensing Arm Development Studio, we are going to create a simple Hello World C project and show you how to specify the base RAM address for a target. For the remainder of this tutorial, we are going to use the Arm Compiler 6 toolchain and our target is a Cortex®-A9 Fixed Virtual Platform, provided with Arm Development Studio.

Prerequisites

Complete [Open Arm® Development Studio for the first time on page 5-39](#)

Procedure

1. Create a new C project:
 - a. If you have no projects open, click **New Project** in the **Project Explorer** view.
 - b. Otherwise, select: **File > New > Project**.
2. Expand the **C/C++** menu, and select **C project**, then click **Next**.
3. In the **C Project** dialog:
 - a. In the **Project name** field, enter HelloWorld.
 - b. Under **Project type**, select **Executable > Hello World ANSI C Project**.
 - c. Under **Toolchains**, select **Arm Compiler 6**.
 - d. Click **Finish**.

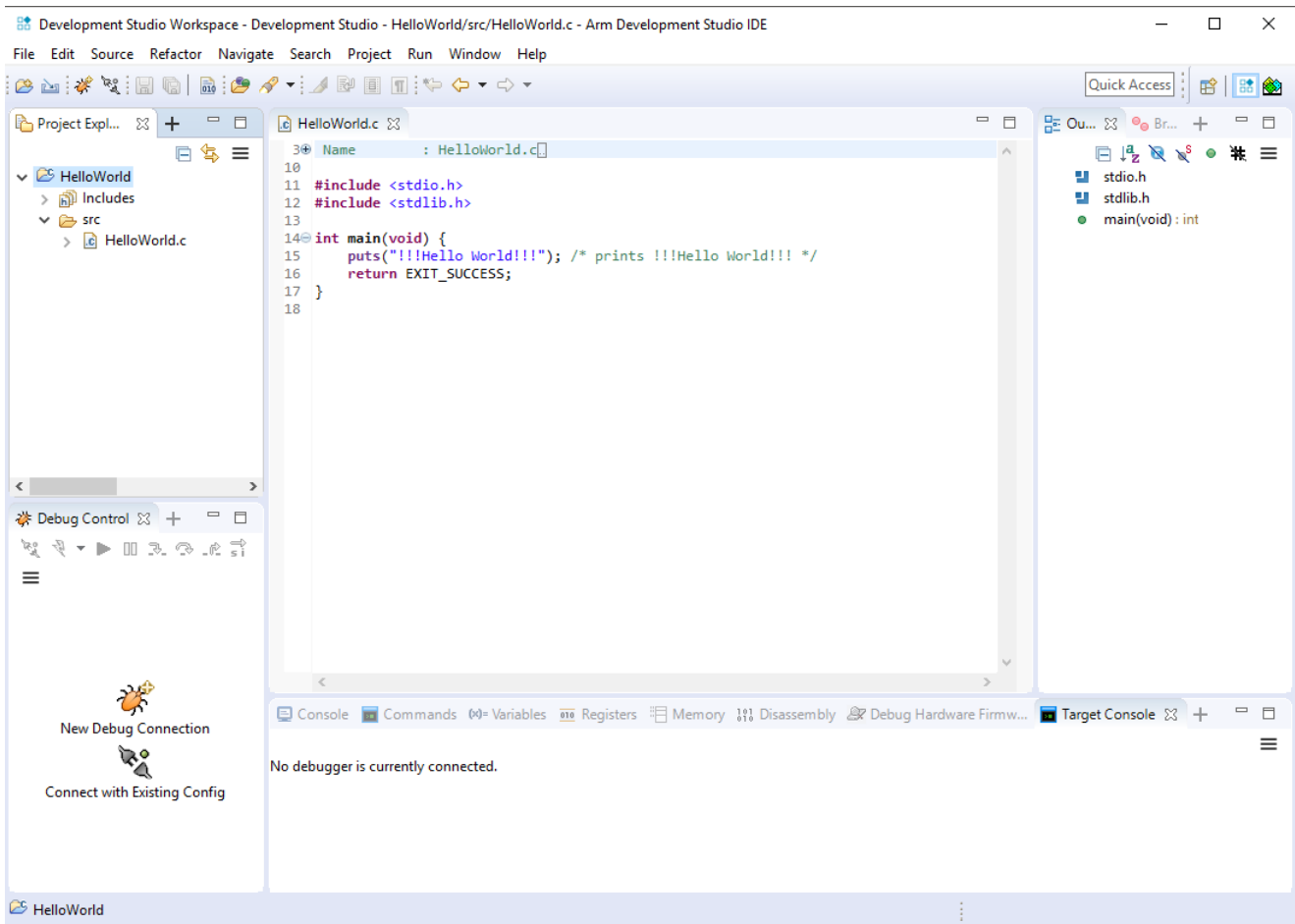


Figure 5-1 Screenshot of the IDE after creating a new project

5.1.3 Configure your project

Before you build the HelloWorld project, you need to tell the linker the RAM base address for your FVP target. This ensures that the application is built and loaded correctly on to your target. You also need to tell Arm Debugger to add debug symbols into the image file, which allows you to debug the image.

Prerequisites

Complete [Create a project in C/C++ on page 5-40](#)

Procedure

1. In the **Project Explorer** view, right-click the HelloWorld project and select **Properties**. The **Properties for HelloWorld** dialog box opens.
2. Add debug symbols into the image file:
 - a. Expand **C/C++ Build**, and select **Build Variables**.
 - b. Set **Configuration** to **Debug [Active]**.
3. Set the linker base RAM address, under **C/C++ Build** select **Settings**:
 - a. Select **Tool Settings > All Tools Settings > Target**.
 - b. From the **Target CPU** dropdown, select **Cortex-A9**.
 - c. From the **Target FPU** dropdown, select **No FPU**.
 - d. Select **Tool Settings > Arm Linker 6 > Image Layout**.
 - e. In the **RO base address** field, enter **0x8000000**.
4. Click **OK**.
5. If you are prompted to rebuild the index, click **Yes**.

5.1.4 Build your project

You can now build your HelloWorld project!

Prerequisites

Complete these tasks:

- [Create a project in C/C++ on page 5-40](#)
- [Configure your project on page 5-41](#)

Procedure

1. In the **Project Explorer** view, right-click the HelloWorld project, and select **Build Project**.

When the project has built, in the **Project Explorer** view, under **Debug**, locate the HelloWorld.axf file.

The .axf file contains the object code and debug symbols that enable Arm Debugger to perform source-level debugging.

Note

Debug symbols are added at build time. You can either specify this manually, using the **-g** option when compiling with Arm Compiler 6, or you can set this to be default behavior. See [Configure your project on page 5-41](#) for details.

5.1.5 Configure your debug session

In Arm Development Studio, you configure a debugging session by creating a debug connection to your target using the **New Debug Connection** wizard.

Depending on your requirements, you can configure a connection to hardware, a Linux application, or a model:

- [Hardware debug connections](#) are for configuring connections to run and debug applications directly on the hardware.
- [Linux application connections](#) are for configuring connections to debug Linux applications running on a target.
- [Model connections](#) are for configuring connections to run and debug applications directly on a CADI-compliant model.

The following example takes you through configuring a **Model Connection** to a Cortex®-A9 Fixed Virtual Platform (FVP), using the project you created in the previous section of this tutorial.

Procedure

1. From the main menu, select **File > New > Model Connection**.
2. In the **Model Connection** dialog box, specify the details of the connection:
 - a. Enter a name for the debug connection, for example **HelloWorld**.
 - b. Select **Use settings from an existing project** option, and select the project that you created and built in the previous section [Build your project on page 5-42](#).
 - c. Click **Next**.
3. In the **Target Selection** dialog box, specify the details of the target:
 - a. Select **Arm FVP (Installed with Arm DS) > VE_Cortex_A9x1**.

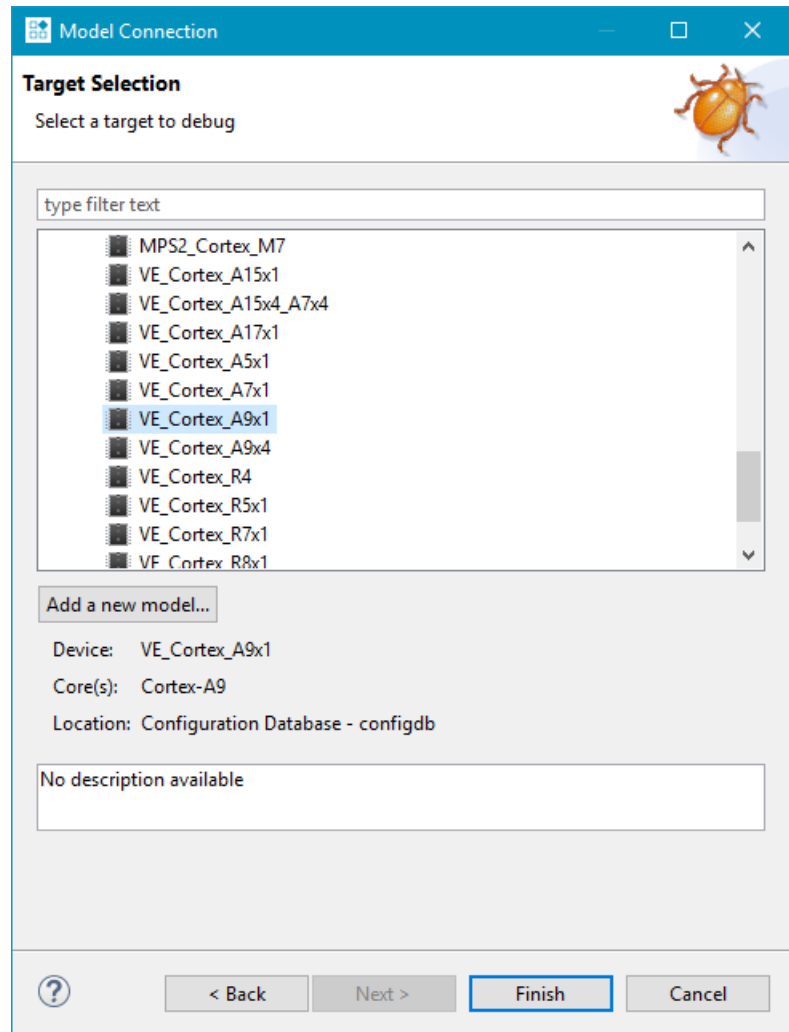


Figure 5-2 Select VE_Cortex_A9x1 model

- b. Click **Finish**.
4. In the **Edit Configuration** dialog, ensure the right target is selected, the appropriate application files are specified, and the debugger knows where to start debugging from:
 - a. Under the **Connection** tab, ensure that **Arm FVP (Installed with Arm DS) > VE_Cortex_A9x1 > Bare Metal Debug > Debug Cortex-A9** is selected.

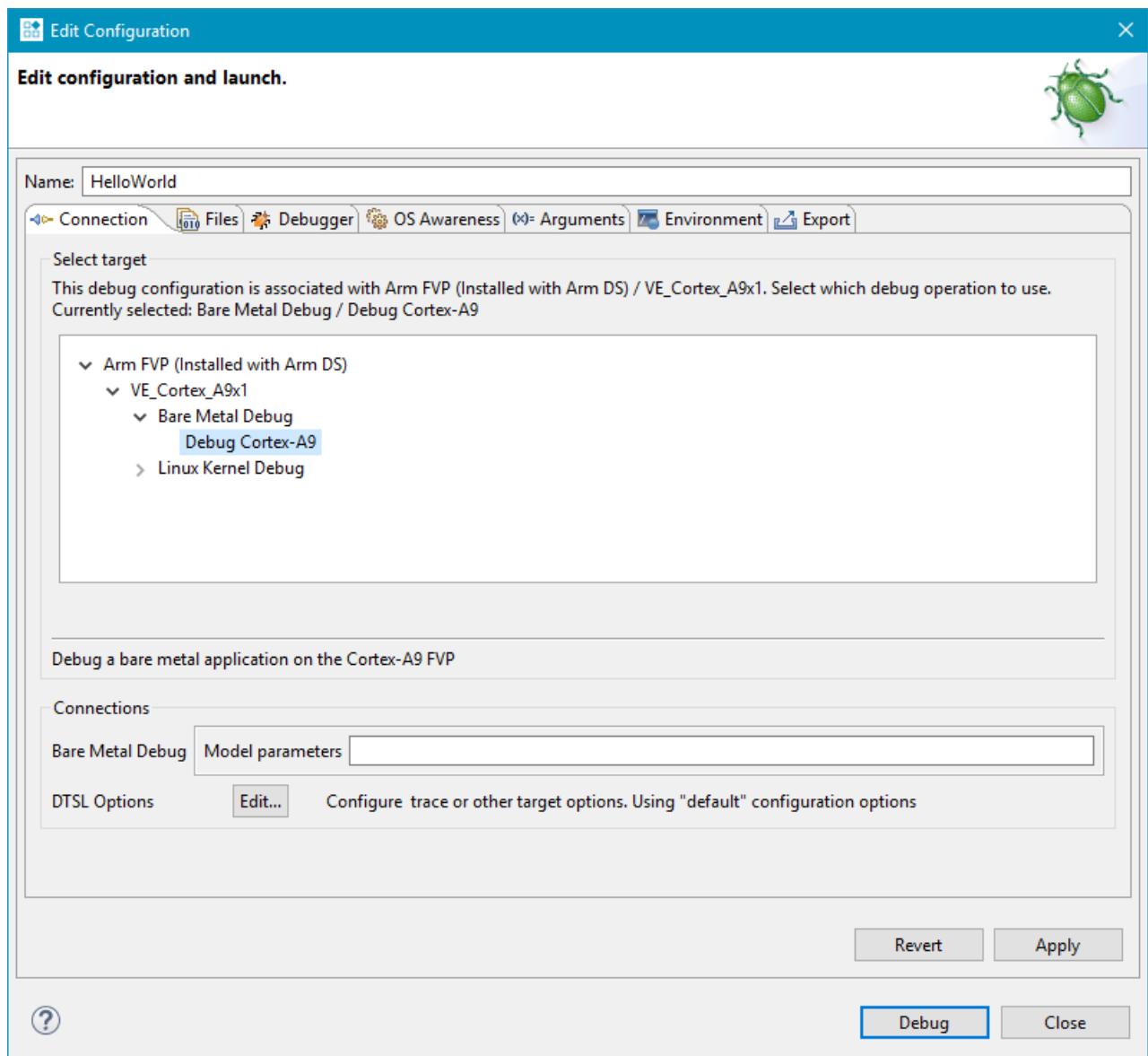


Figure 5-3 Edit configuration Connection tab

- b. In the **Files** tab, select **Target Configuration** > **Application on host to download** > **Workspace**.

Edit configuration and launch.

❌ A file has been specified to be downloaded to the target, which will require the core to be stopped, but "Connect Only" has also been specified on the Debugger tab.

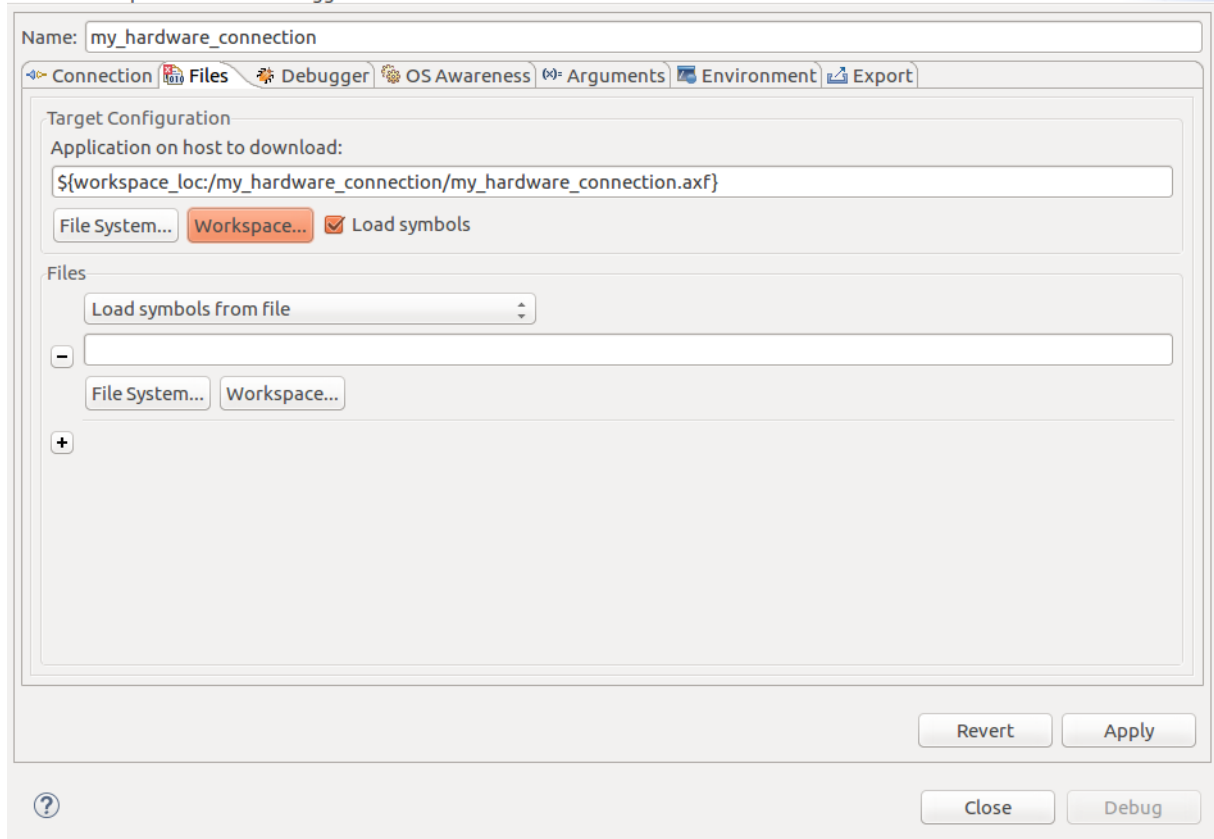


Figure 5-4 Edit configuration Files tab

- c. Click and expand the **HelloWorld** project and from the **Debug** folder, select **HelloWorld.axf** and click **OK**.

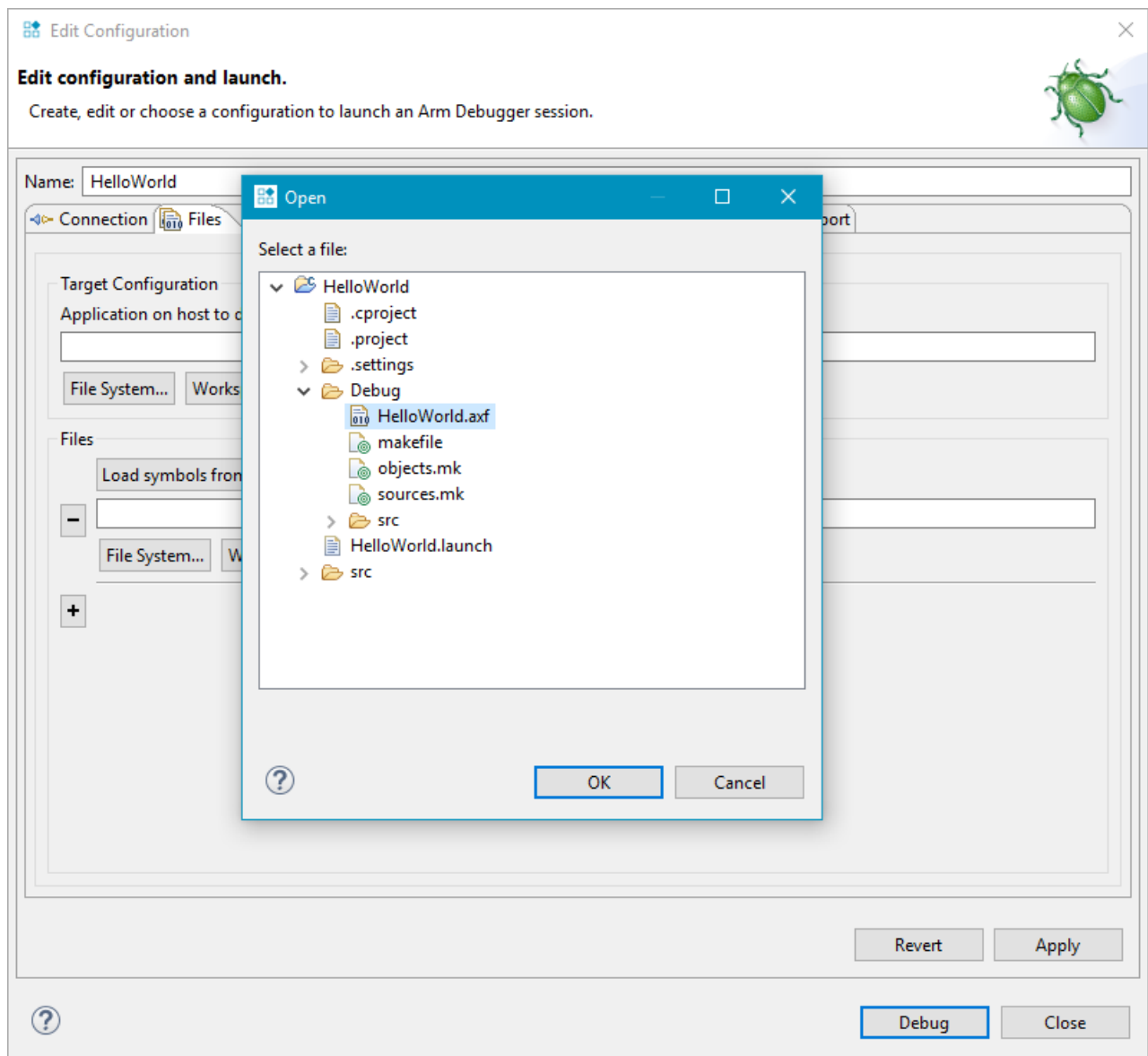


Figure 5-5 Select helloworld.axf file

- d. In the **Debugger** tab, select **Debug from symbol**.

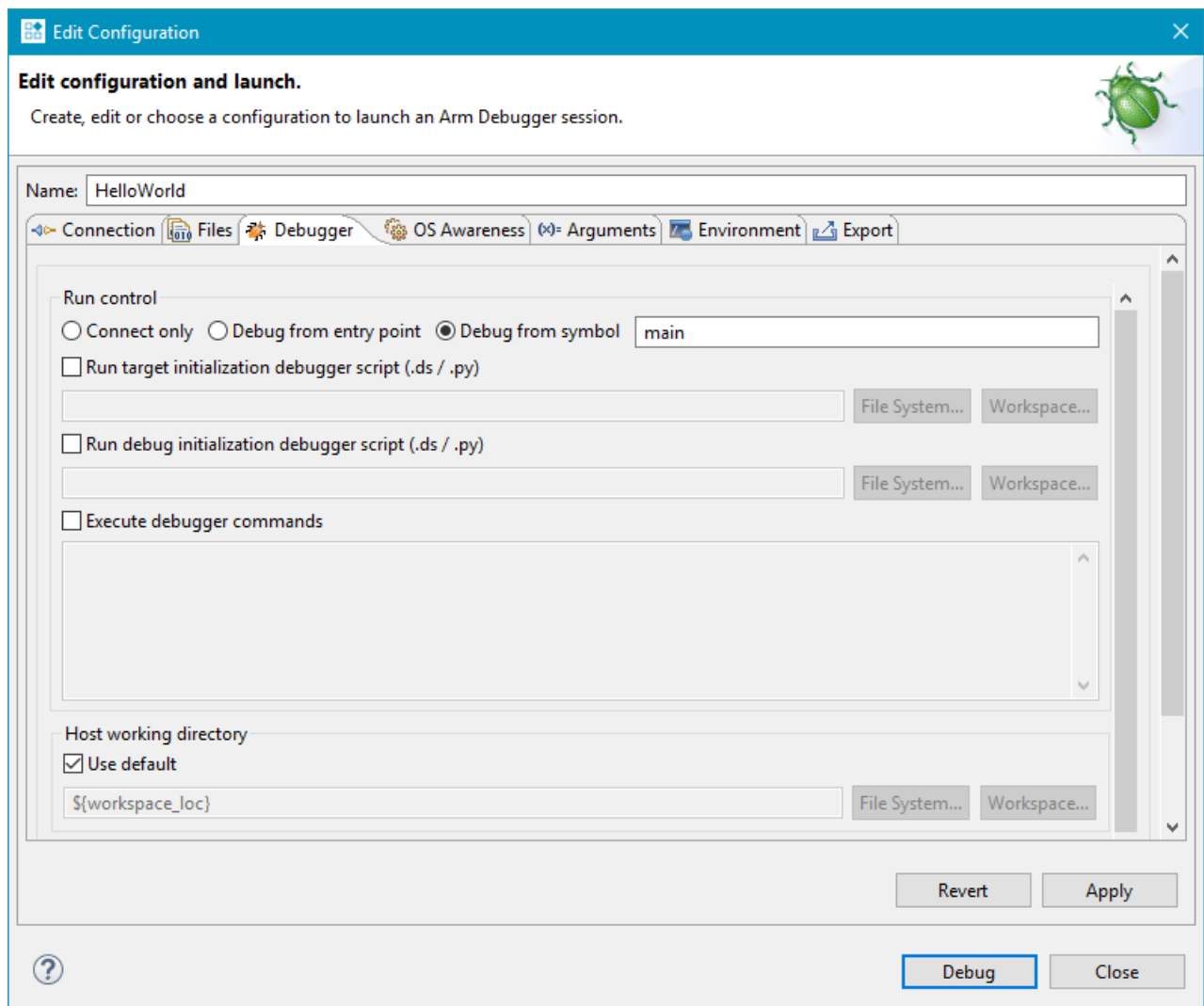


Figure 5-6 Debug from symbol main

5. Click **Debug** to load the application on the target, and load the debug information into the debugger.
Arm Development Studio connects to the model and displays the connection status in the **Debug Control** view.

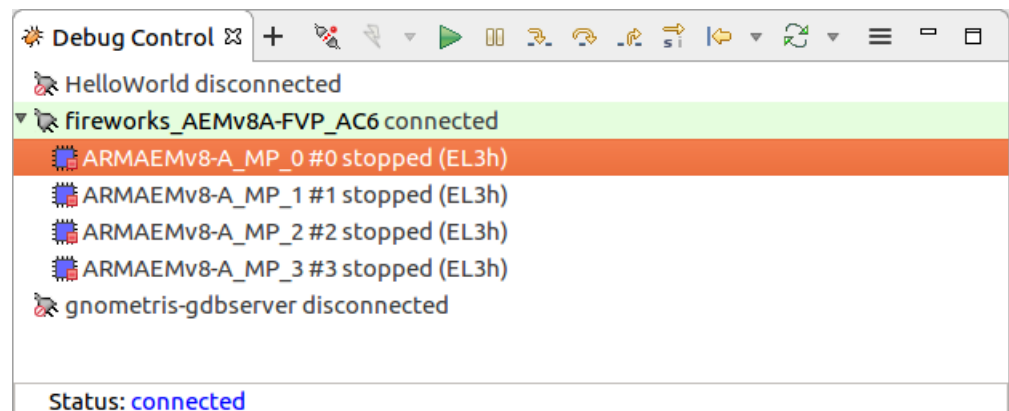
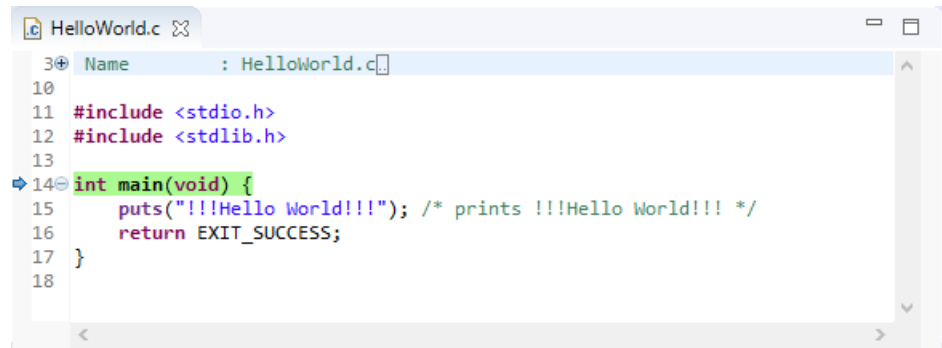


Figure 5-7 Debug Control View

The application loads on the target, and stops at the `main()` function, ready to run.



```

3  Name      : HelloWorld.c
10
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 int main(void) {
15     puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
16     return EXIT_SUCCESS;
17 }
18

```

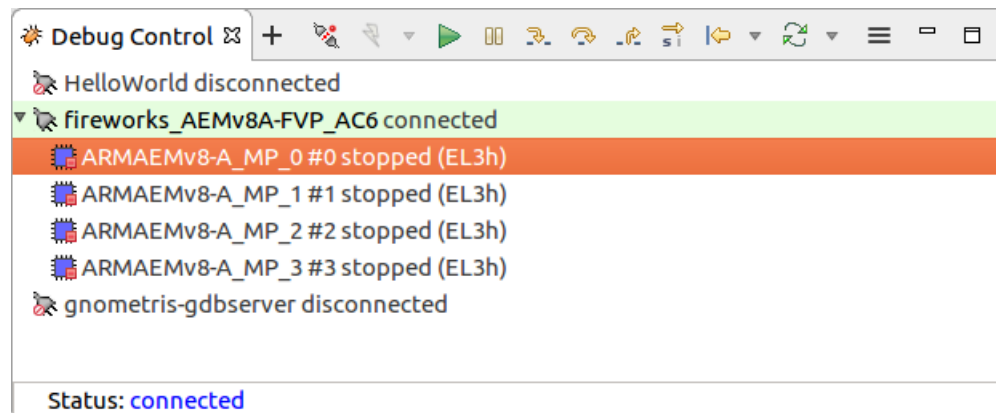
Figure 5-8 main () in code editor







5.1.6 Application debug with Arm Debugger

Now that you have created a debug configuration and the application is loaded on the target, it is time to start debugging and stepping through your application.

Running and stepping through the application

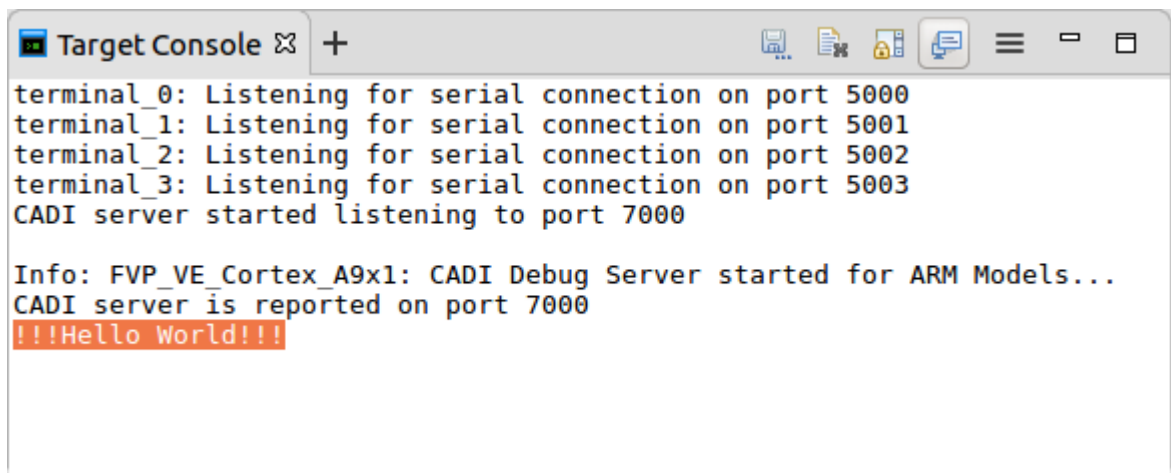
Use the controls provided in the **Debug Control** view to debug your application. By default, these controls do source level stepping.



-  - Click to continue running the application after loading it on the target.
-  - Click to interrupt or pause executing code.
-  - Click to step through the code.
-  - Click to step over a source line.
-  - Click to step out.
-  - This is a toggle. Click this to toggle between stepping instructions and stepping source code. This applies to the above step controls.

Other views display information relevant to the debug connection

- **Target Console** view displays the application output.



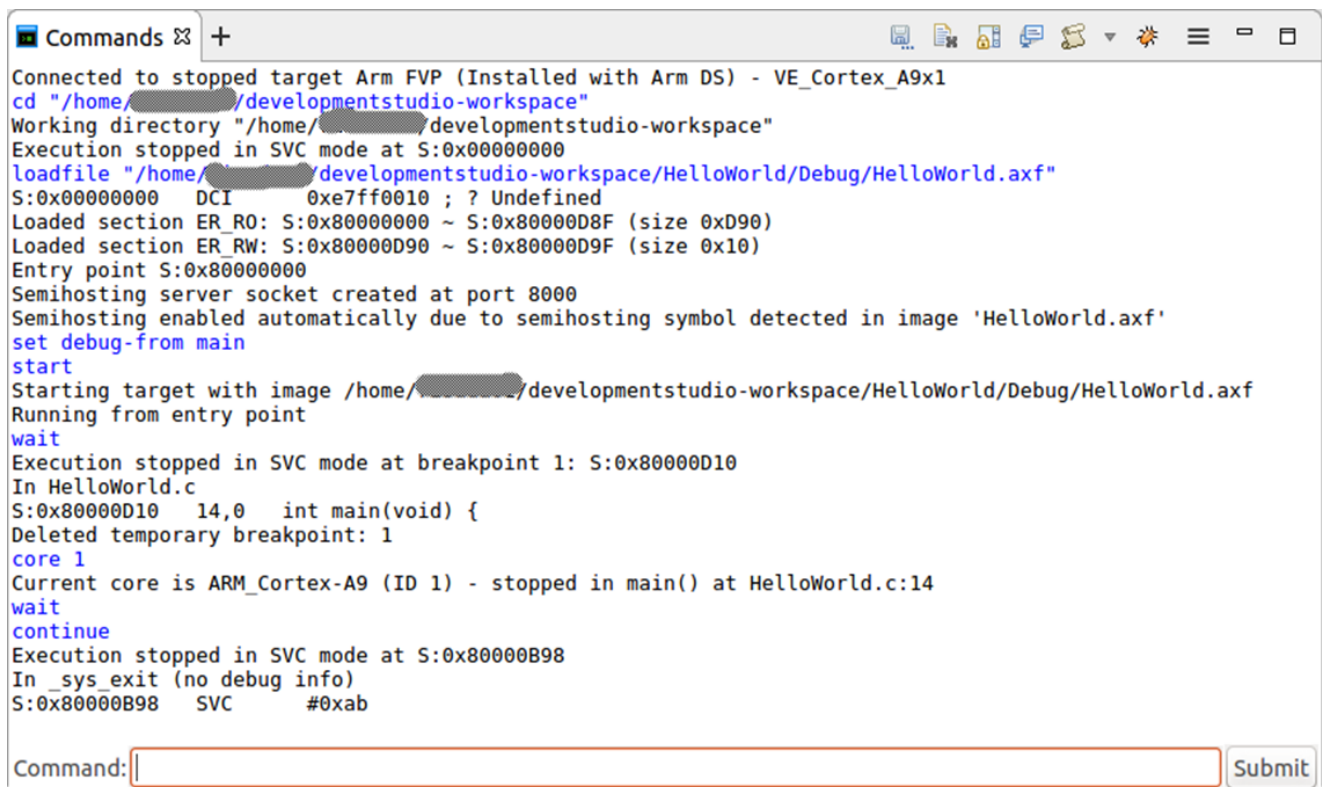
```

Target Console
terminal_0: Listening for serial connection on port 5000
terminal_1: Listening for serial connection on port 5001
terminal_2: Listening for serial connection on port 5002
terminal_3: Listening for serial connection on port 5003
CADI server started listening to port 7000

Info: FVP_VE_Cortex_A9x1: CADI Debug Server started for ARM Models...
CADI server is reported on port 7000
!!!Hello World!!!
  
```

Figure 5-9 Target console output

- **Commands** view displays messages output by the debugger. Also use this view to enter Arm Debugger commands.



```

Commands
Connected to stopped target Arm FVP (Installed with Arm DS) - VE_Cortex_A9x1
cd "/home/developmentstudio-workspace"
Working directory "/home/developmentstudio-workspace"
Execution stopped in SVC mode at S:0x00000000
loadfile "/home/developmentstudio-workspace/HelloWorld/Debug/HelloWorld.axf"
S:0x00000000 DCI 0xe7ff0010 ; ? Undefined
Loaded section ER_R0: S:0x80000000 ~ S:0x8000008F (size 0xD0)
Loaded section ER_RW: S:0x80000090 ~ S:0x8000009F (size 0x10)
Entry point S:0x80000000
Semihosting server socket created at port 8000
Semihosting enabled automatically due to semihosting symbol detected in image 'HelloWorld.axf'
set debug-from main
start
Starting target with image /home/developmentstudio-workspace/HelloWorld/Debug/HelloWorld.axf
Running from entry point
wait
Execution stopped in SVC mode at breakpoint 1: S:0x800000D10
In HelloWorld.c
S:0x800000D10 14,0 int main(void) {
Deleted temporary breakpoint: 1
core 1
Current core is ARM_Cortex-A9 (ID 1) - stopped in main() at HelloWorld.c:14
wait
continue
Execution stopped in SVC mode at S:0x800000B98
In _sys_exit (no debug info)
S:0x800000B98 SVC #0xab

Command:  Submit
  
```

Figure 5-10 Commands view

- C/C++ Editor view shows the active C, C++, or Makefile. The view updates when you edit these files.

```

HelloWorld.c
3+ Name      : HelloWorld.c
10
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 int main(void) {
15     puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
16     return EXIT_SUCCESS;
17 }
18

```

Figure 5-11 Code Editor view

- **Disassembly** view shows the built program as assembly instructions, and their memory location.

Address	Opcode	Disassembly
S:0x80000B8A	2000	MOV _S r0, #0
S:0x80000B8C	BD1C	POP {r2-r4, pc}
S:0x80000B8E	4620	MOV r0, r4
S:0x80000B90	BD1C	POP {r2-r4, pc}
S:0x80000B92	0000	MOV _S r0, r0
S:0x80000B94	4901	LDR r1, [pc, #4] ; [0x80000B9C] = 0x20026
S:0x80000B96	2018	MOV _S r0, #0x18
S:0x80000B98	0FAB	SVC #0xab
S:0x80000B9A	E7FE	B _sys_exit+6 ; 0x80000B9A
S:0x80000B9C	00020026	DCD 0x00020026
S:0x80000BA0	4770	BX lr
S:0x80000BA2	4770	BX lr
S:0x80000BA4	468C	MOV r12, r1
S:0x80000BA6	6843	LDR r3, [r0, #4]
S:0x80000BA8	2B00	CMP r3, #0
S:0x80000BAA	BF18	IT NE
S:0x80000BAC	428B	CMP r3, r1
S:0x80000BAE	BF38	IT CC
S:0x80000BB0	4618	MOV r0, r3

Figure 5-12 Disassembly view

➡ indicates the location in the code where your program is stopped. In this case, it is at the `main()` function.

- **Memory** view shows how the code is represented in the target memory. For example, to view how the string `Hello World` from the application is represented in memory:
 1. Open the **Memory** view.
 2. In the **Address** field, enter `&main` and press **Enter** on your keyboard. The view displays the contents of the target's memory.
 3. Select and highlight the words `Hello World`.

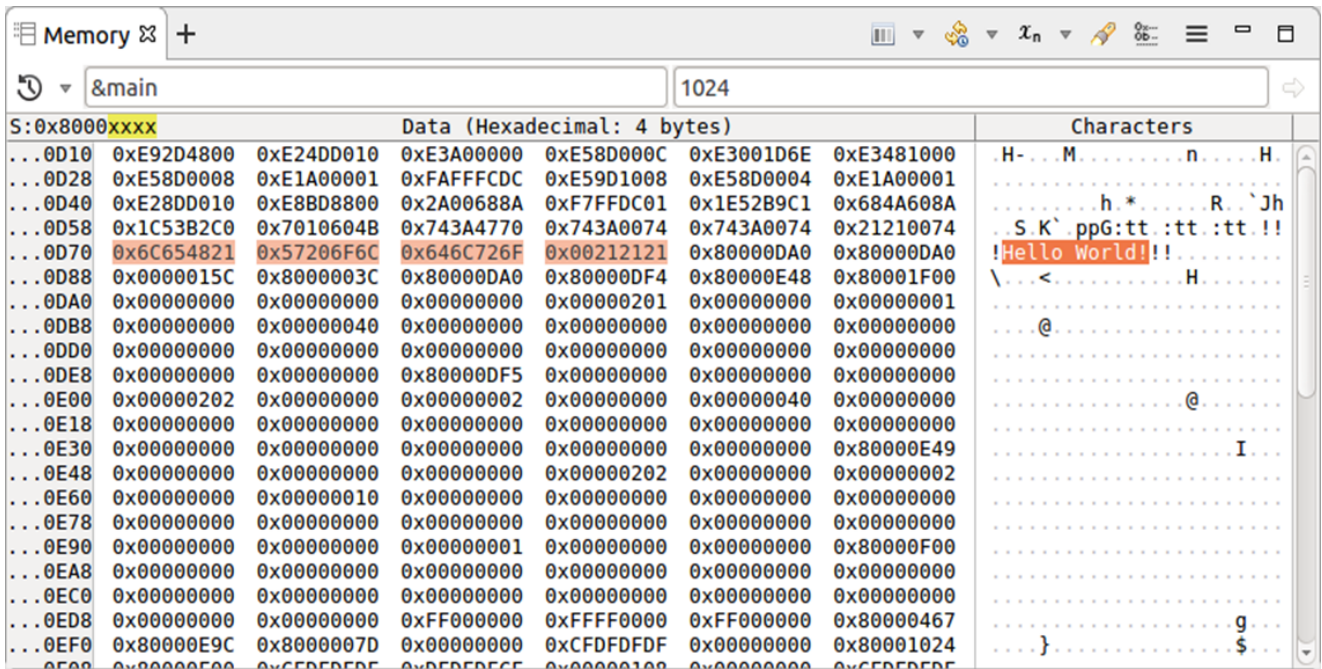


Figure 5-13 Memory view

In the above example, the **Memory** view displays the hexadecimal values for the code and the ASCII character encoding of the memory values, which enable you to view the details of the code.

After completing your debug activities, you can *disconnect the target* on page 5-51.

5.1.7 Disconnecting from a target

To disconnect from a target, you can use either the **Debug Control** or the **Commands** view.

- If you are using the **Debug Control** view, click **Disconnect from Target** on the toolbar.

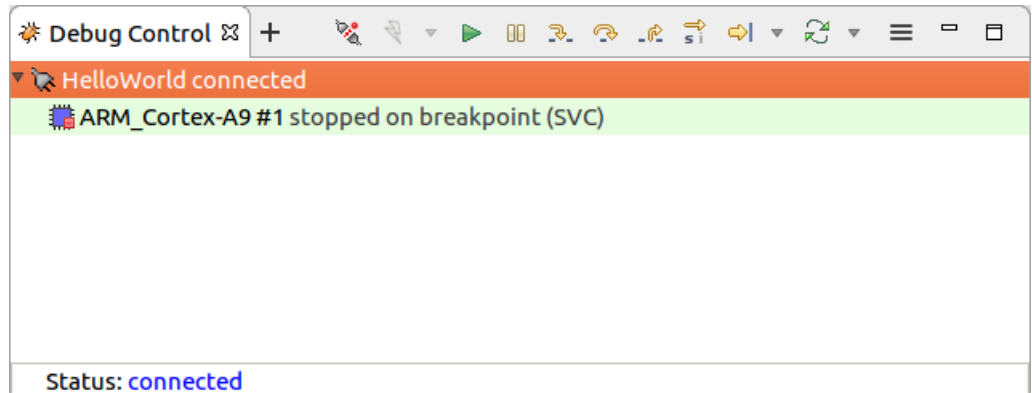


Figure 5-14 Disconnecting from a target using the Debug Control view

- If you are using the **Commands** view, enter **quit** in the **Command** field, then click **Submit**.

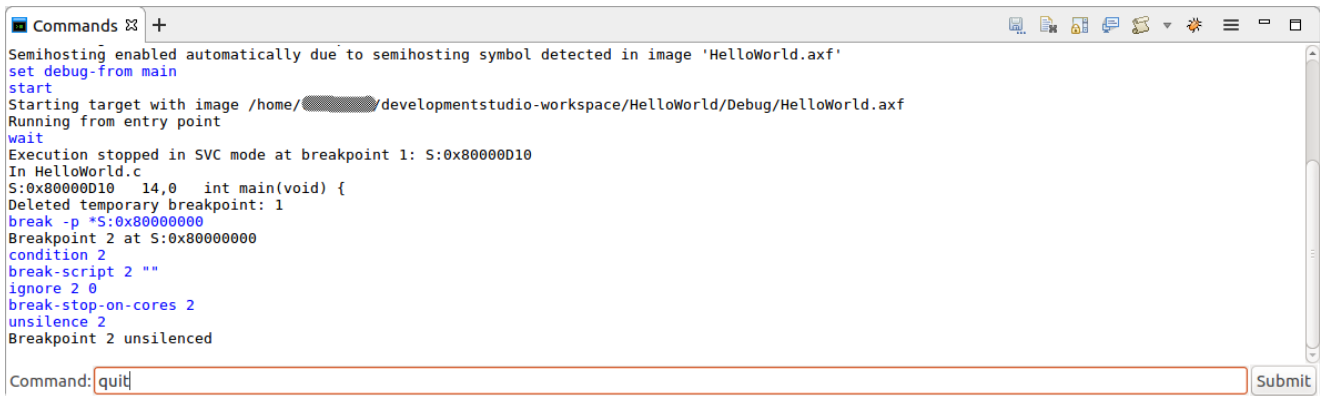


Figure 5-15 Disconnecting from a target using the Commands view

The disconnection process ensures that the state of the target does not change, except for the following case:

- Any downloads to the target are canceled and stopped.
- Any breakpoints are cleared on the target, but are maintained in Arm Development Studio.
- The DAP (Debug Access Port) is powered down.
- Debug bits in the DSC (Debug Status Control) register are cleared.

If a trace capture session is in progress, trace data continues to be captured even after Arm Development Studio has disconnected from the target.

Appendix A

Terminology and Shortcuts

Supplementary information for new users of Arm Development Studio.

It contains the following sections:

- [A.1 Terminology on page Appx-A-54.](#)
- [A.2 Keyboard shortcuts on page Appx-A-56.](#)

A.1 Terminology

Arm Development Studio documentation uses a range of terms. These are listed below.

Device

A component on a target that contains the application that you want to debug.

Dialog box

A small page that contains tabs, panels, and editable fields which prompt you to enter information.

Editor

A view that enables you to view and modify the content of a file, for example source files. The tabs in the editor area show files that are currently open for editing.

Flash Program

A term used to describe the storing of data on a flash device.

IDE

The Integrated Development Environment. A window that contains perspectives, menus, and toolbars. This is the main development environment where you can manage individual projects, associated sub-folders, and source files. Each window is linked to one workspace.

Panel

A small area in a dialog box or tab to group editable fields.

Perspective

Perspectives define the layout of your selected views and editors in Eclipse. They also have their own associated menus and toolbars.

Project

A group of related files and folders in Eclipse.

Resource

A generic term used to describe a project, file, folder, or a combination of these.

Send To

A term used to describe sending a file to a target.

Tab

A small overlay page that contains panels and editable fields within a dialog box to group related information. Clicking on a tab brings it to the top.

Target

A development platform on a printed circuit board or a software model that emulates the expected behavior of Arm hardware.

View

Views provide related information, for a specific function, corresponding to the active file in the editor. They also have their own associated menus and toolbars.

Wizard

A group of dialog boxes to guide you through common tasks. For example, creating new files and projects.

Workspace

An area on your file system used to store files and folders related to your projects.

A.2 Keyboard shortcuts

A list of the most common keyboard shortcuts available for use with Arm Development Studio.

F3

Click an assembly instruction and press F3 to see help information about the instruction.

F10

Press F10 to access the main menu. You can then navigate the main menu using the arrow keys.

Alt+F4

Exit Arm Development Studio.

Alt+Left arrow

Go back in navigation history.

Alt+Right arrow

Go forward in navigation history.

Ctrl+Semicolon

In the Arm assembler editor, add comment markers to a selected block of code in the active file.

Ctrl+Home

Move the editor focus to the beginning of the code.

Ctrl+End

Move the editor focus to the end of the code.

Ctrl+B

Build all projects in the workspace that have changed since the last build.

Ctrl+F

Open the Find or Find/Replace dialog box to search through the code in the active editor. Some editors are read-only and therefore disable this functionality.

Ctrl+F4

Close the active file in the editor view.

Ctrl+F6

Cycle through open files in the editor view.

Ctrl+F7

Cycle through available views.

Ctrl+F8

Cycle through available perspectives.

Ctrl+F10

Use with the arrow keys to access the drop-down menu.

Ctrl+L

Move to a specified line in the active file.

Ctrl+Q

Move to the last edited position in the active file.

Ctrl+Space

Auto-complete selected functions in editors.

Shift+F10

Use with the arrow keys to access the context menu.

Ctrl+Shift+F

Activate the code style settings in the **Preferences** dialog box and apply them to the active file.

Ctrl+Shift+L

Open a small page with a list of all keyboard shortcuts.

Ctrl+Shift+R

Open the **Open resource** dialog box.

Ctrl+Shift+T

Open the **Open Type** dialog box.

Ctrl+Shift+/

In the C/C++ editor, add comment markers to the start and end of a selected block of code in the active file.