Arm® Development Studio

Version 2020.1

Getting Started Guide

arm

101469_2010_00_en

Arm® Development Studio

Getting Started Guide

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
1901-00	15 July 2019	Non-Confidential	Updated document for Arm® Development Studio 2019.0-1
1910-00	1 November 2019	Non-Confidential	Updated document for Arm® Development Studio 2019.1
2000-00	20 March 2020	Non-Confidential	Updated document for Arm® Development Studio 2020.0
2000-01	3 July 2020	Non-Confidential	Documentation update 1 for Arm® Development Studio 2020.0
2010-00	28 October 2020	Non-Confidential	Updated document for Arm® Development Studio 2020.1

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

developer.arm.com

Contents
Arm® Development Studio Getting Started Guide

List of Figures
Arm® Development Studio Getting Started Guide

List of Tables
Arm® Development Studio Getting Started Guide

Preface

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

It contains the following:

- *About this book* on page 10.

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_2010_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® Documentation*.
- *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.

Arm 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

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.

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 behavior 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
- Arm DSTREAM-PT
- Arm DSTREAM-HT
- Keil® ULINK™ 2
- Keil ULINKpro™
- Keil ULINKpro D

Supported third-party debug adapters include:

- ST-Link
- Cadence virtual debug
- FTDI MPSSE JTAG

  ——— **Note** ———

  If you are using the FTDI MPSSE JTAG adapter on Linux, the OS automatically installs an incorrect driver when you connect this adapter. For details on how to fix this issue, see *Troubleshooting: FTDI probe incompatible driver error* in the *Arm Development Studio User Guide*.

  ———————————

- USB-Blaster II

  ——— **Note** ———

  If you are using the USB-Blaster debug units, Arm Debugger can connect to Arria V SoC, Arria 10 SoC, Cyclone V SoC and Stratix 10 boards. To enable the connections, ensure that the environment variable `QUARTUS_ROOTDIR` is set and contains the path to the 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 Quartus tools installation path. For example, `~/<quartus_tools_installation_directory>/qprogrammer`.

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

  ———————————

Debug connections

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

Supported debug connections include:

- CADI (debug interface for models)
- Iris interface for models
- Ethernet to gdbserver
- CMSIS-DAP

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.

Related information

Troubleshooting: FTDI probe incompatible driver error

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

Arm Streamline 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.

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

Related information

Streamline documentation

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.

Related information

Graphics Analyzer documentation

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

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 Arm Streamline.

Host platform requirements

Arm Development Studio supports the following host platforms:

- Windows 10
- 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. For a list of supported debug units, see *Debug Probes* on page 1-16.

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
Arm 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 ULINKpro and ULINKpro 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 2-24.
- *2.3.2 Using the command line* on page 2-24.

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. This opens the Arm Development Studio setup wizard.
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 Example using msiexec

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

Install Arm Development Studio on Linux using the installation package provided on the Arm developer website.

Note

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

Prerequisites

Download the Linux installation package from the *Arm Developer website*.

Procedure

- Run `armds-<version>.sh` and follow the on-screen instructions.

  Note

  During the installation, Arm Development Studio automatically runs a dependency check and produces a list of missing libraries. You can safely continue with the installation. Arm recommends that you install these libraries before using Arm Development Studio.

  You can find more details and a full list of required libraries in *Additional Linux libraries on page 2-27*.

  Note

  Arm recommends that you run the post install setup scripts during the installation process.

Next Steps

To use the post install setup scripts after installation, with root privileges, run:

`run_post_install_for_Arm_DS_IDE_<version>.sh`

This script is in the install directory.

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

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. This script is in `<install_location>/sw/dependency_check`.

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-gnueabihf-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. 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.

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`
- `libxpcom.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

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 one of the following:
  - The license server address and port number.
  - The license file.
- To obtain an evaluation license, you need an Arm account.

Procedure

1. Add your license:

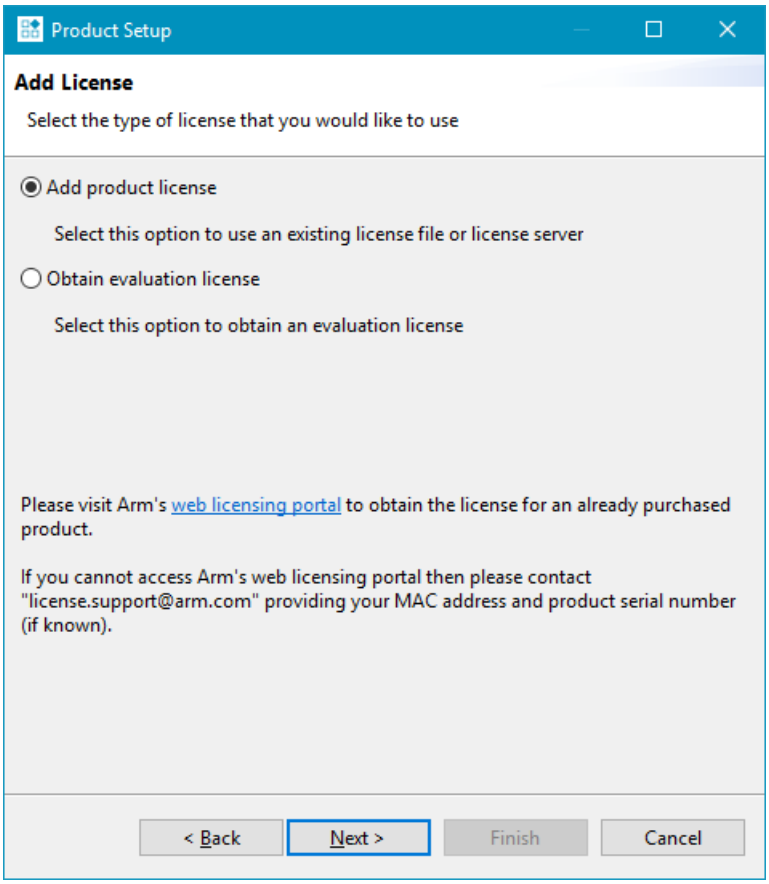


Figure 3-1 Product Setup dialog box when you first open Arm Development Studio.

   - For a license server, select **Add product license**, and click **Next**. Enter the license server address and port number, in the form `<port number>@<server address>`. Click **Next**.
   - For a license file, select **Add product license**, and click **Next**. Click **Browse…** and select the license file. 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.
2. 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

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.

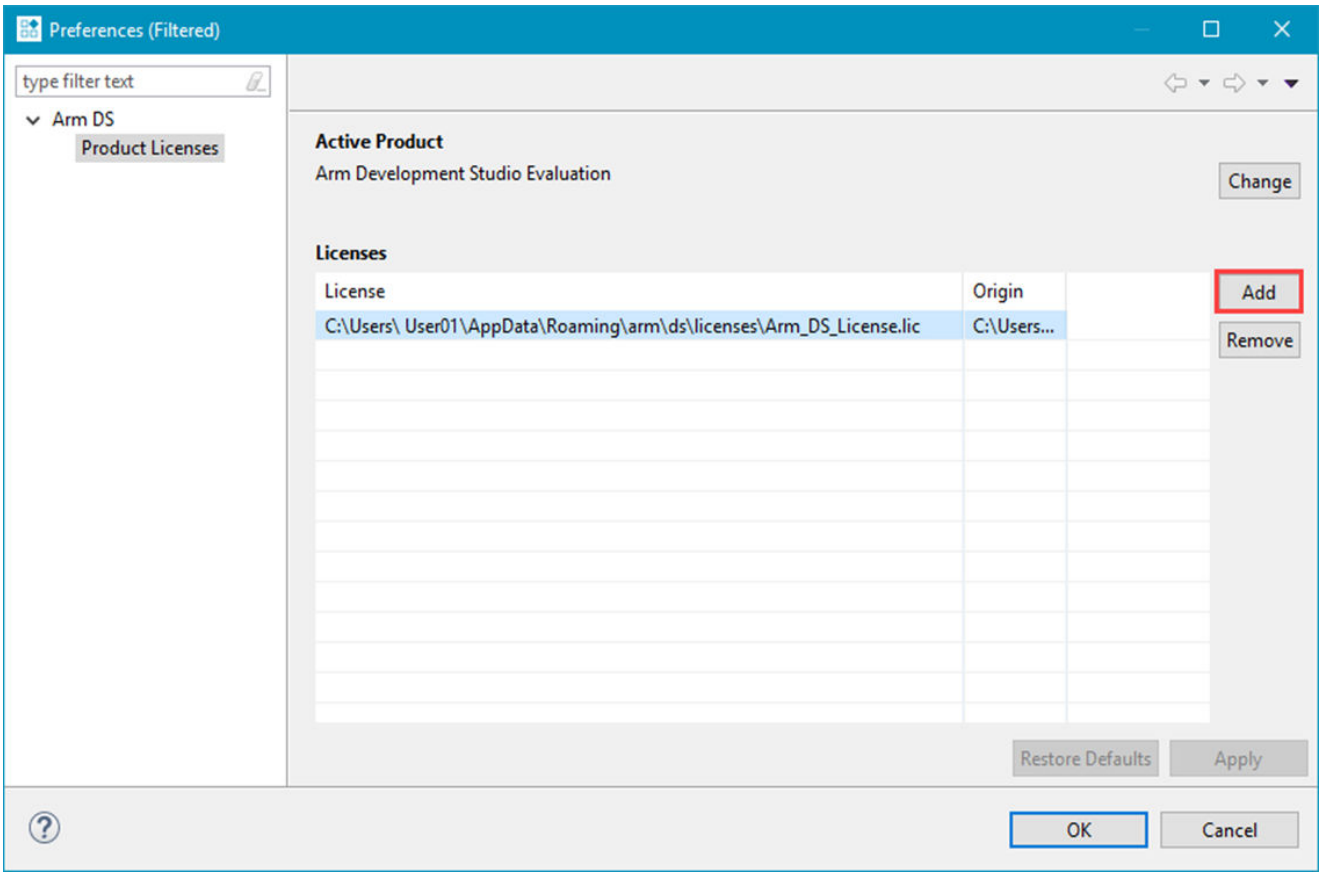


Figure 3-2 Adding a license in preferences dialog box.

3. Follow the steps in *Using Product Setup to add a license* on page 3-29 to add your license.

Related tasks

3.1 Using Product Setup to add a license on page 3-29

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**.

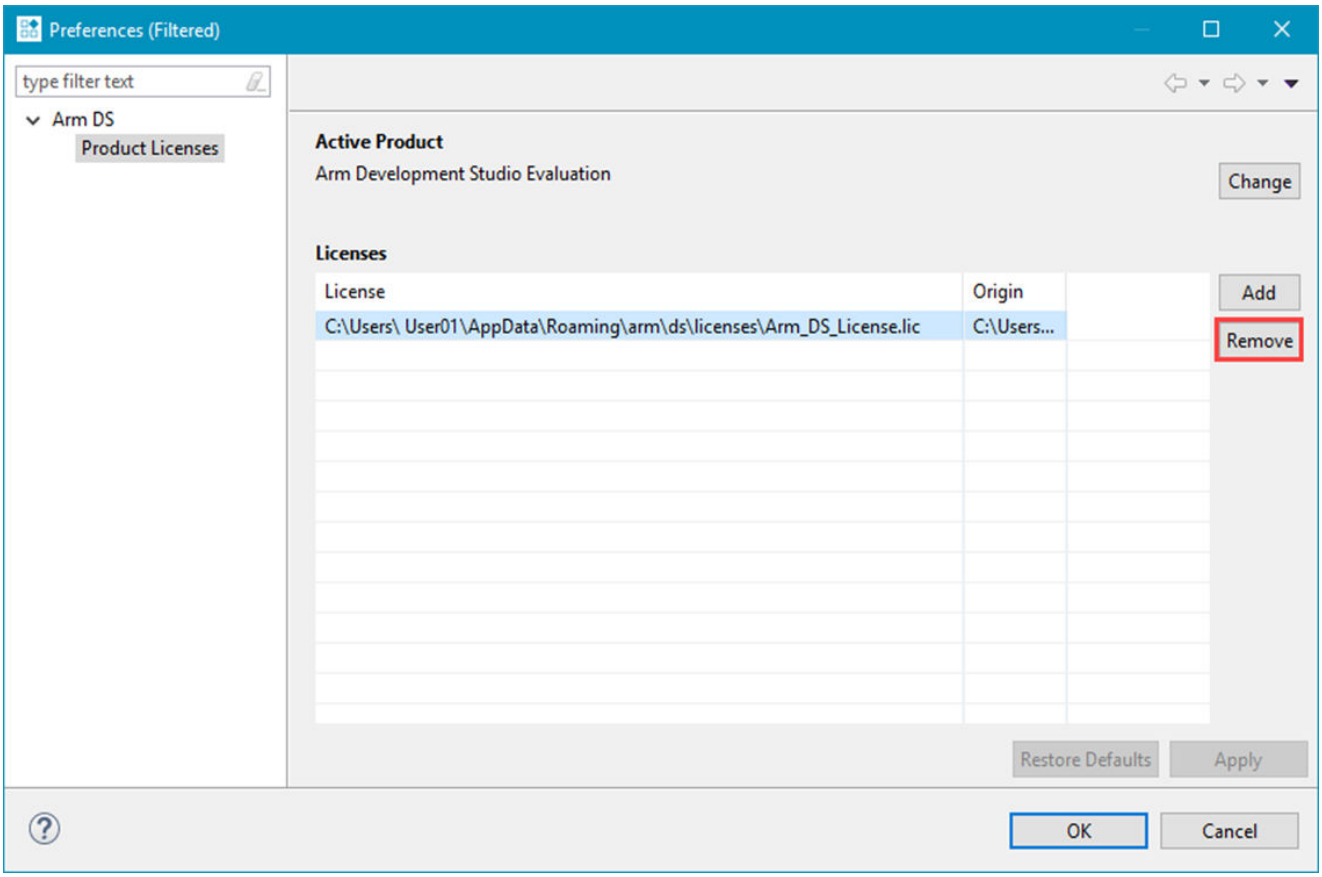


Figure 3-3 Deleting a license in preferences dialog box.

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** > **Perspective** > **Open Perspective**.

It contains the following sections:

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.

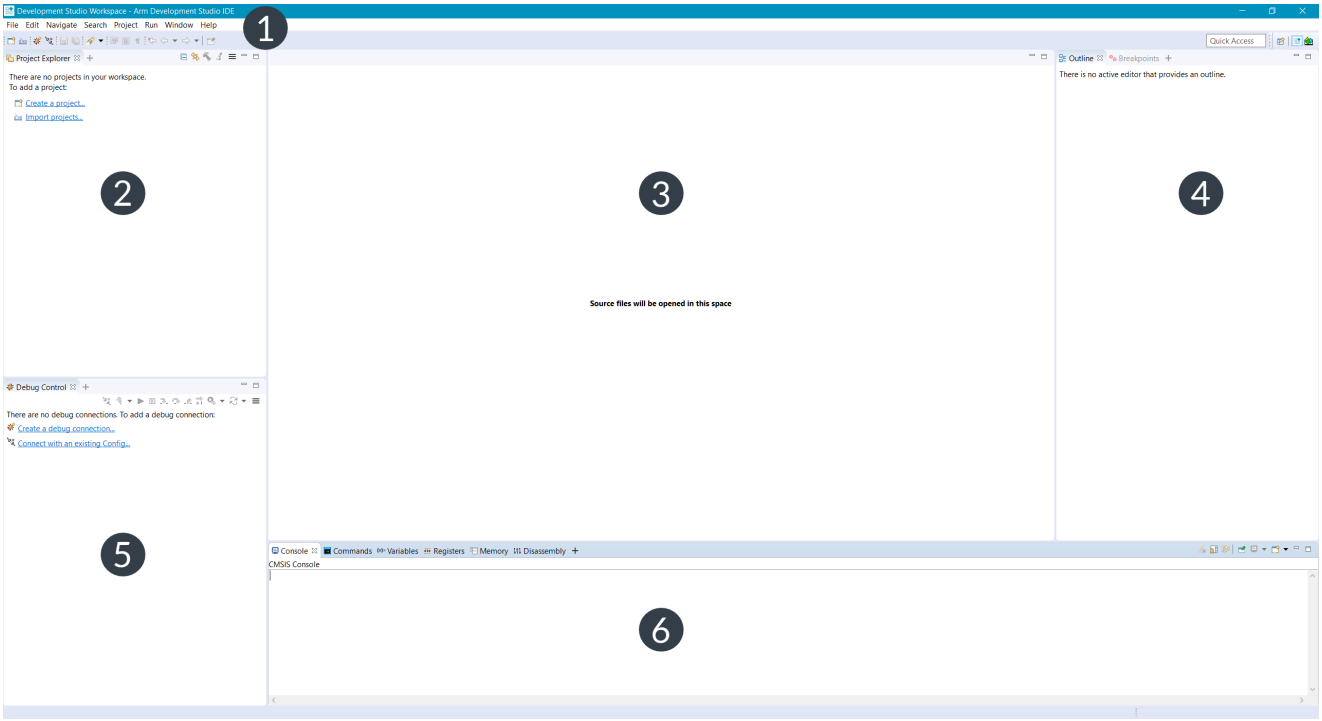


Figure 4-1 IDE in the Development Studio perspective.

1. 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.
2. Project Explorer - Use this view to create, build, and manage your projects.
3. Debug Control - Use this view to create and control debug connections.
4. 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.
5. Debug views - Views specific to Arm Debugger. Multiple views can be stacked in an area, creating tabs.
6. Area containing additional debug views. All areas are fully customizable to contain any desired 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 Personalize your development environment

Arm Development Studio Integrated Development Environment (IDE) has many settings, called **Preferences**, that are available for you to adjust and change. Use these **Preferences** to adapt the IDE to best support your own personal development style.

When you launch Arm Development Studio for the first time, the **Preferences Wizard** takes you through the process of setting up the IDE.

This wizard presents the most commonly changed **Preferences** to customize for your requirements. These include specifying the start-up workspace location, selecting a theme, and tweaking the code editing format.

Note

- If you have upgraded from a previous version of Arm Development Studio and had your workspace preferences already set up, your preferences remain the same.
- You can click **Apply and Close** at any point during your wizard. The **Preferences Wizard** applies changes up to where you have modified the options and leaves the rest of the settings as default.
- There are more IDE configuration options in the **Preferences** dialog which allow you to make further in-depth changes to your IDE settings. For example, extra code formatting and syntax highlighting options. To open the **Preferences** dialog, from the main menu, select **Window** > **Preferences**.
- You can click **Skip** and ignore the **Preferences Wizard** and return to the wizard later to make changes. To restart the wizard later, in the **Preferences** dialog, select **Arm DS** > **General** > **Start Preferences Wizard**.

Related tasks

4.4 Language settings on page 4-39

Related references

Chapter 2 Installation on page 2-21

Chapter 3 Licensing Arm® Development Studio on page 3-28

4.3 Using the IDE on page 4-35

4.3 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:

4.3.1 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.

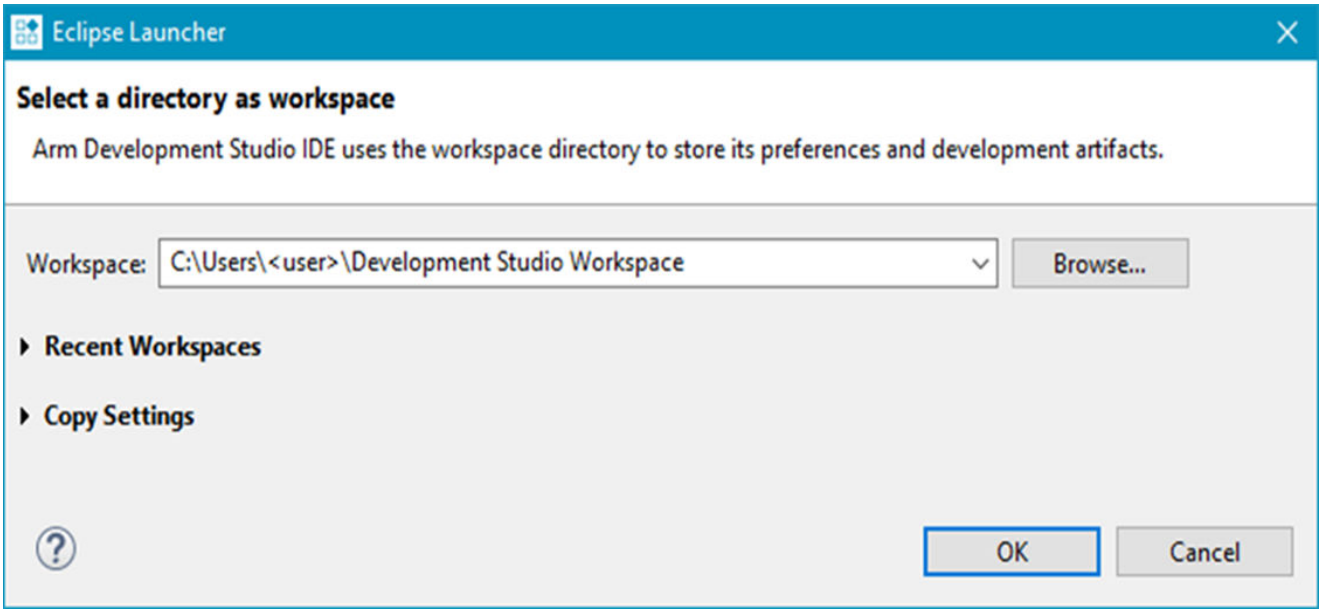


Figure 4-2 Workspace Launcher dialog box

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

Arm Development Studio relaunches in the new workspace.

4.3.2 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**.

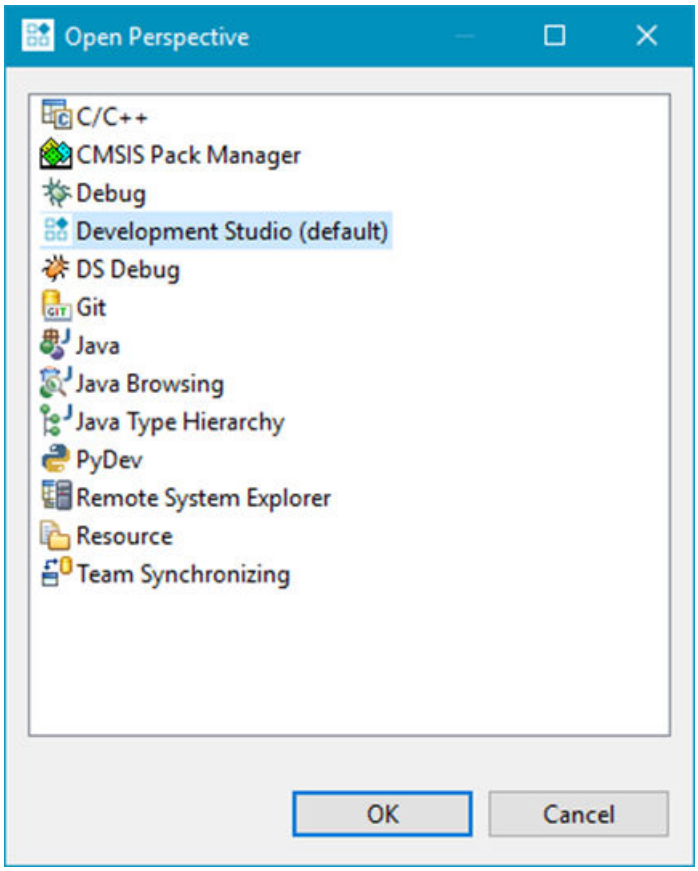


Figure 4-3 Open Perspective dialog box

Your perspective opens in the workspace.

Related information

Arm Debugger perspectives and views

4.3.3 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.

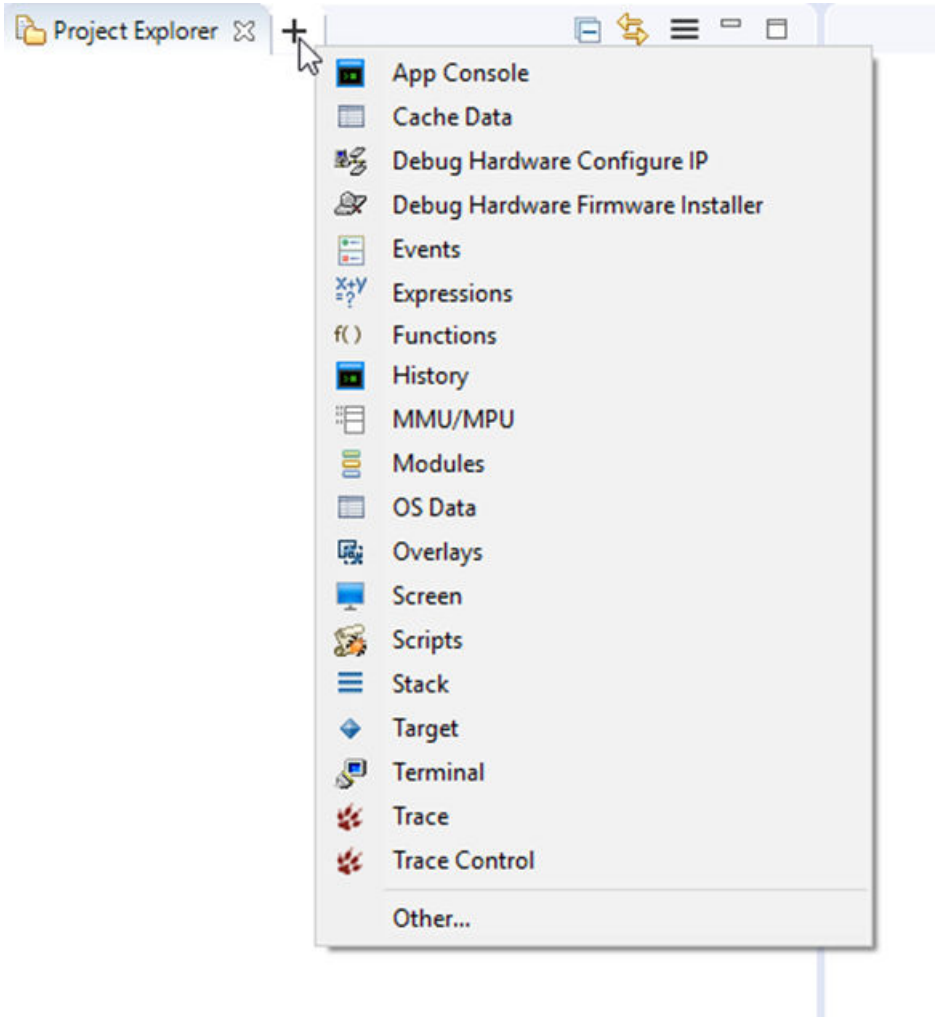


Figure 4-4 Adding a view in an area

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

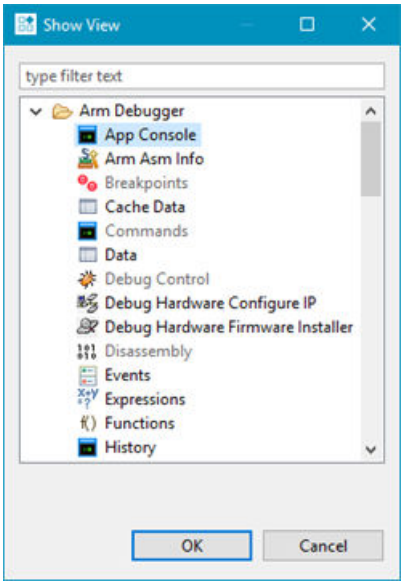


Figure 4-5 Adding a view in Arm Development Studio

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

4.4 Language settings

Only Japanese language packs are currently supported by Arm Development Studio. These language packs are installed with Arm Development Studio.

Procedure

- Launch the IDE in Japanese using one of the following methods:
  - If your operating system locale is set as Japanese, the IDE automatically displays the translated features.
  - If your operating system locale is not set as Japanese, you must specify the `-nl` command-line argument when launching the IDE:

    ```
    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

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

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 2-22.

Prerequisites

- Download and install Arm Development Studio, for either:
  - Linux: *Installing on Linux* on page 2-26
  - Windows: *Installing on Windows* on page 2-24
- 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 box 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 4-33*, 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 5-41*
- Ensure you are in the **Development Studio** Perspective. This is the default perspective when Arm Development Studio is first opened. To return to it, click the **Development Studio** button in the top right corner.

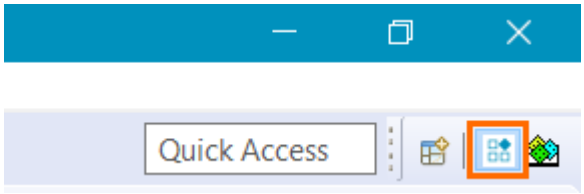


Figure 5-1 Screenshot highlighting the button for the Development Studio Perspective.

Procedure

1. To create a new C project, select: **File** > **New** > **Project...**.
2. Expand the **C/C++** menu, and select **C project**, then click **Next**.
3. In the **C Project** dialog box:
   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**.

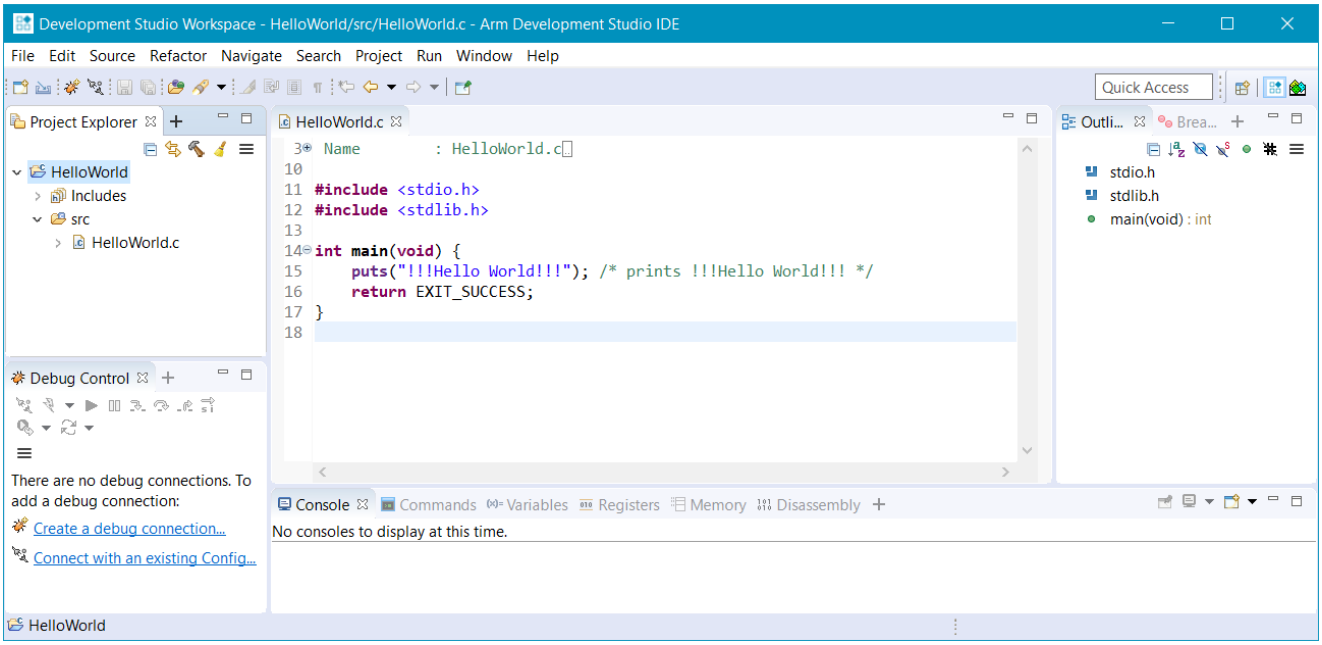


Figure 5-2 The IDE after creating a new project

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 5-42

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. In the **Tool Settings** tab, select **All Tools Settings** > **Target**.
   b. From the **Target CPU** dropdown, select **Cortex-A53 AArch64**.
   c. From the **Target FPU** dropdown, select **Armv8 (Neon)**.
   d. Select **Arm Linker 6** > **Image Layout**.
   e. In the **RO base address** field, enter `0x80000000`.
4. Click **Apply and Close**.
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 5-42
- *Configure your project* on page 5-43

Procedure

- 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 5-43* 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. Create a `.ds` script to disable semihosting by Arm Debugger:
   a. From the main menu, select **File** > **New** > **Other...**.
   b. In the **New** dialog box, select **Arm Debugger** > **Arm Debugger Script** and click **Next**.
   c. Click **Workspace…** and select the **HelloWorld** project as the location for this script. Click **OK**.
   d. In the **File Name** field, name this script `use_model_semihosting` and click **Finish**. The empty script opens in the **Editor** window.
   e. Add the following code to the script and press **Ctrl** + **S** to save:

```
set semihosting enabled off
```

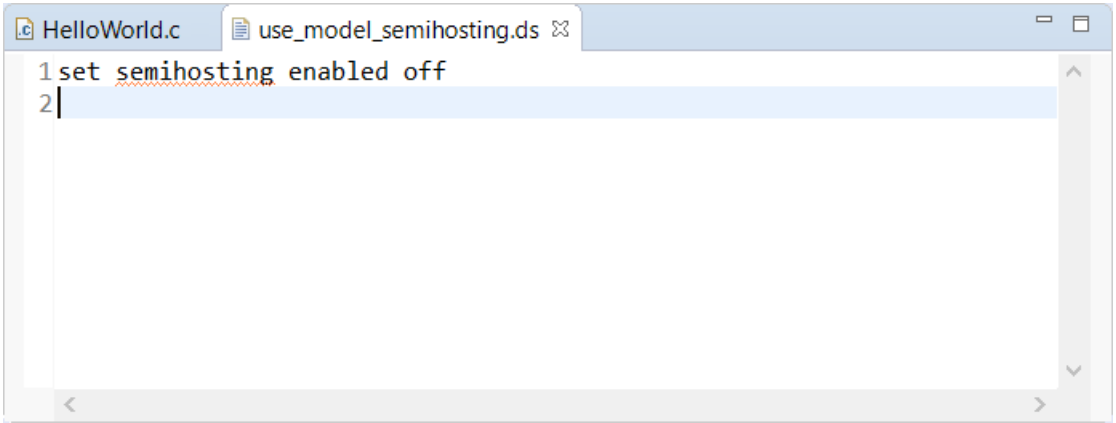


Figure 5-3 Editor window with semihosting script.

2. From the main menu, select **File** > **New** > **Model Connection**.
3. In the **Model Connection** dialog box, specify the details of the connection:

a. Enter a name for the debug connection, for example **HelloWorld_FVP**.
b. Select **Associate debug connection with an existing project**, and select the project that you created and built in the previous section *Build your project* on page 5-43.
c. Click **Next**.

4. In the **Target Selection** dialog box, specify the details of the target:
   a. Select **Arm FVP (Installed with Arm DS)** > **Base_A53x1**.

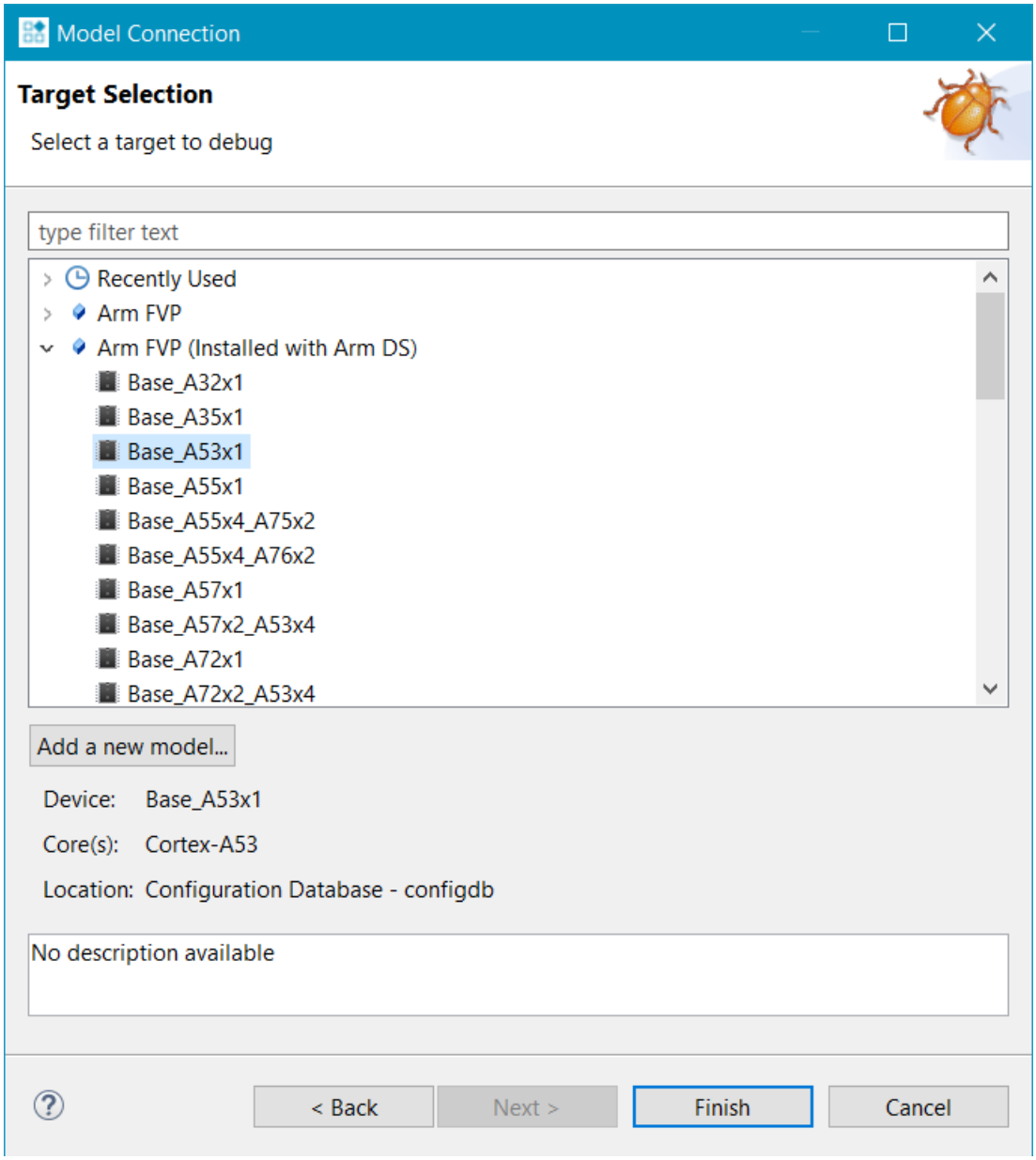


Figure 5-4 Select Base_A53x1 model

   b. Click **Finish**.

5. In the **Edit Configuration** dialog box, 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)** > **Base_A53x1** > **Bare Metal Debug** > **Debug Cortex-A53** is selected.
   b. Under **Bare Metal Debug**, in the **Model parameters** field, add the following parameter:

```
-C bp.secure_memory=false
```

This parameter disables the TZC-400 TrustZone memory controller included in the Base_A53x1 FVP. By default, the memory controller refuses all accesses to DRAM memory.

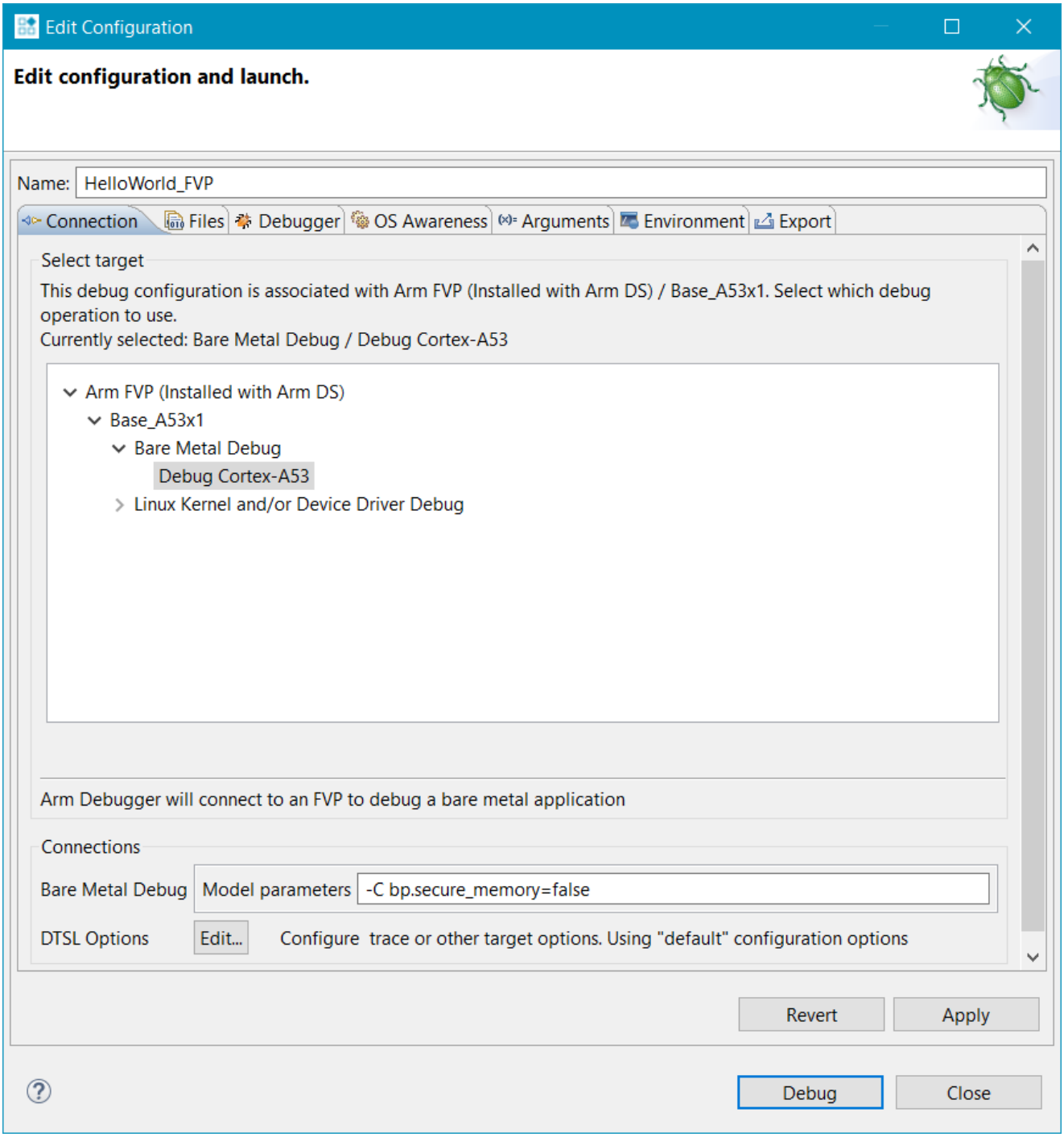


Figure 5-5 Edit configuration Connection tab

c. In the **Files** tab, select **Target Configuration** > **Application on host to download** > **Workspace**.
d. Click and expand the **HelloWorld** project and from the **Debug** folder, select `HelloWorld.axf` and click **OK**.

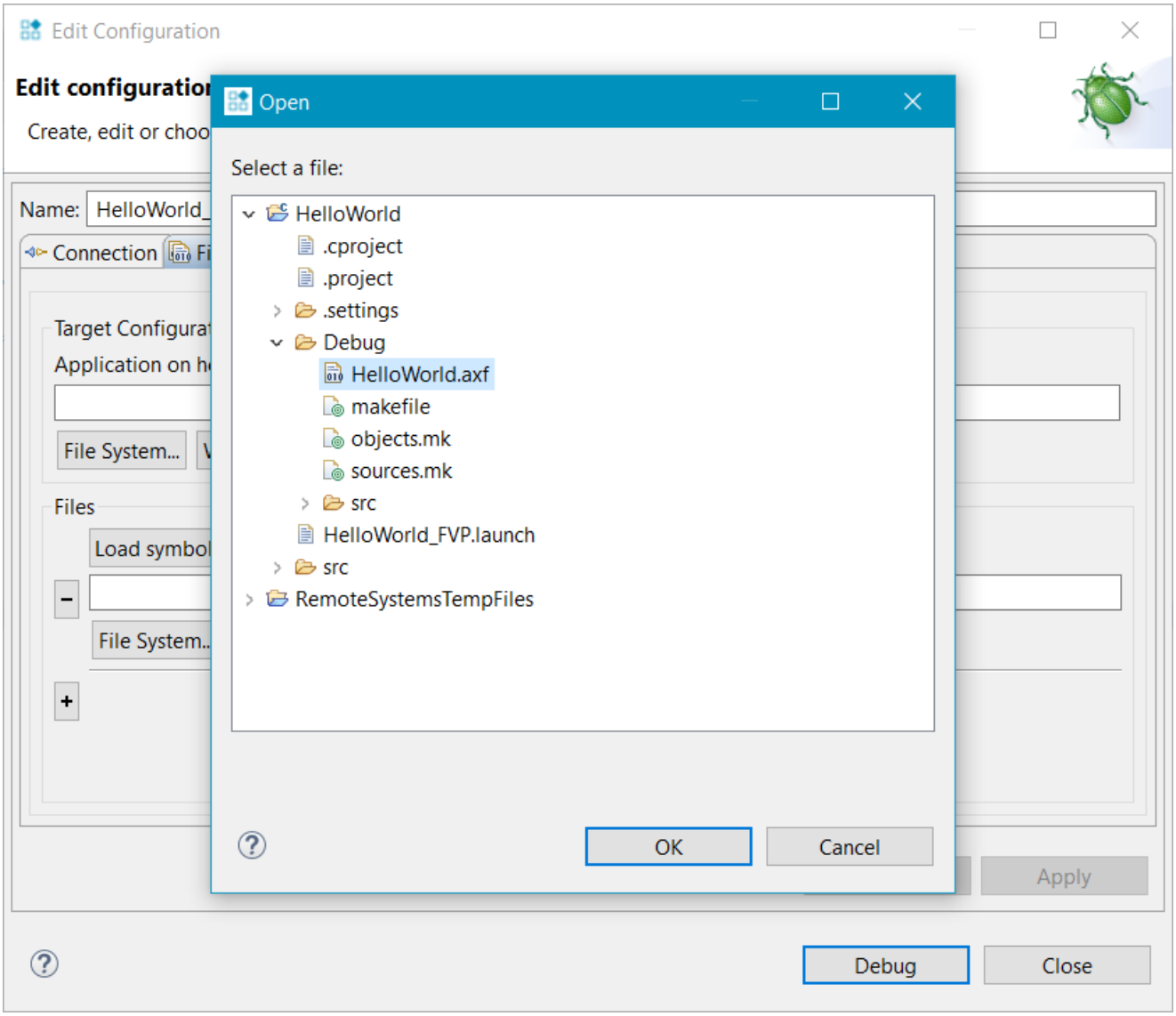


Figure 5-6 Select helloworld.axf file

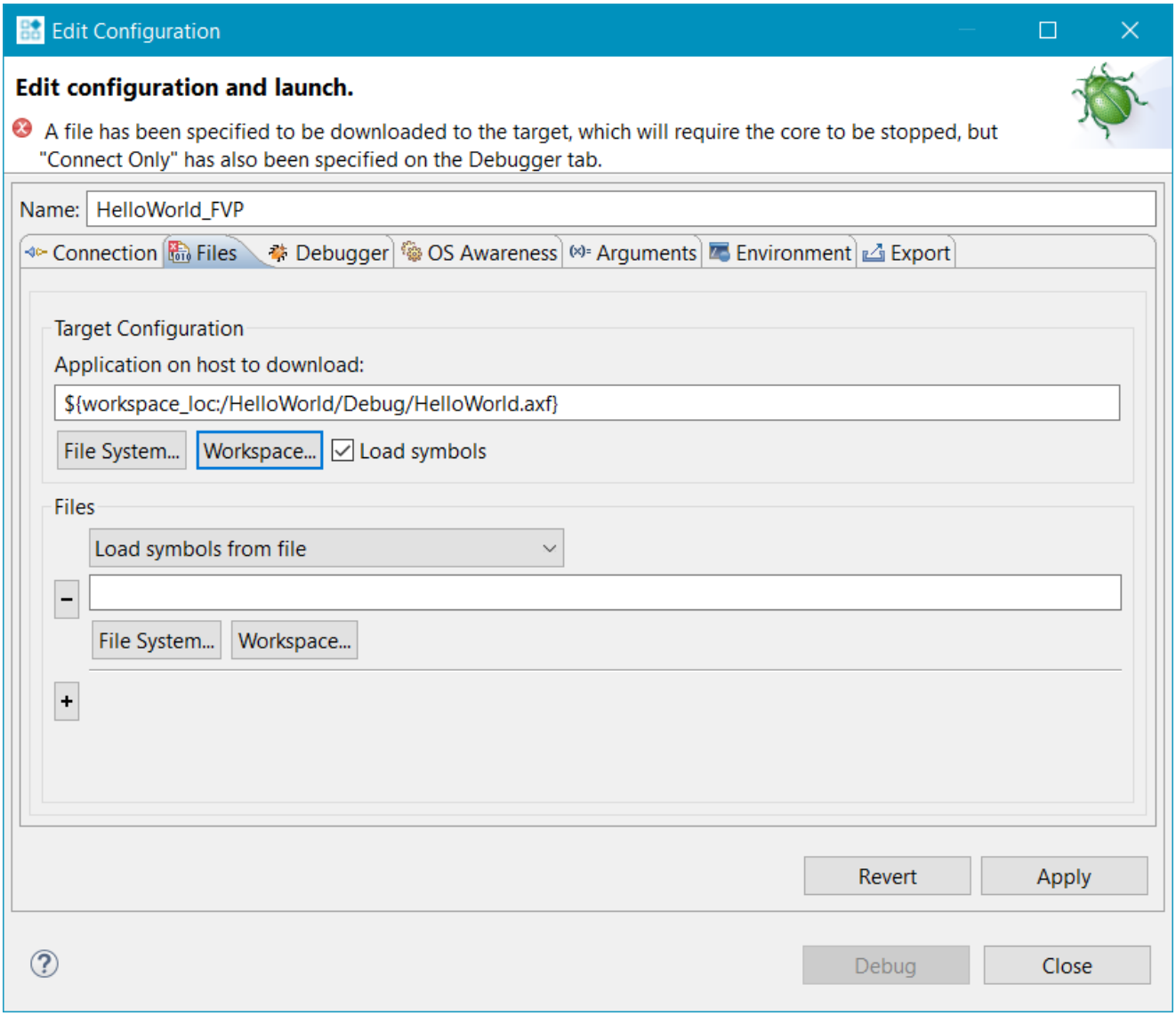


Figure 5-7 Edit configuration Files tab

e. In the **Debugger** tab, select **Debug from symbol**.
f. Enable **Run target initialization debugger script (.ds/.py)** and click **Workspace…**.
g. Select the `use_model_semihosting.ds` script and click **OK**.

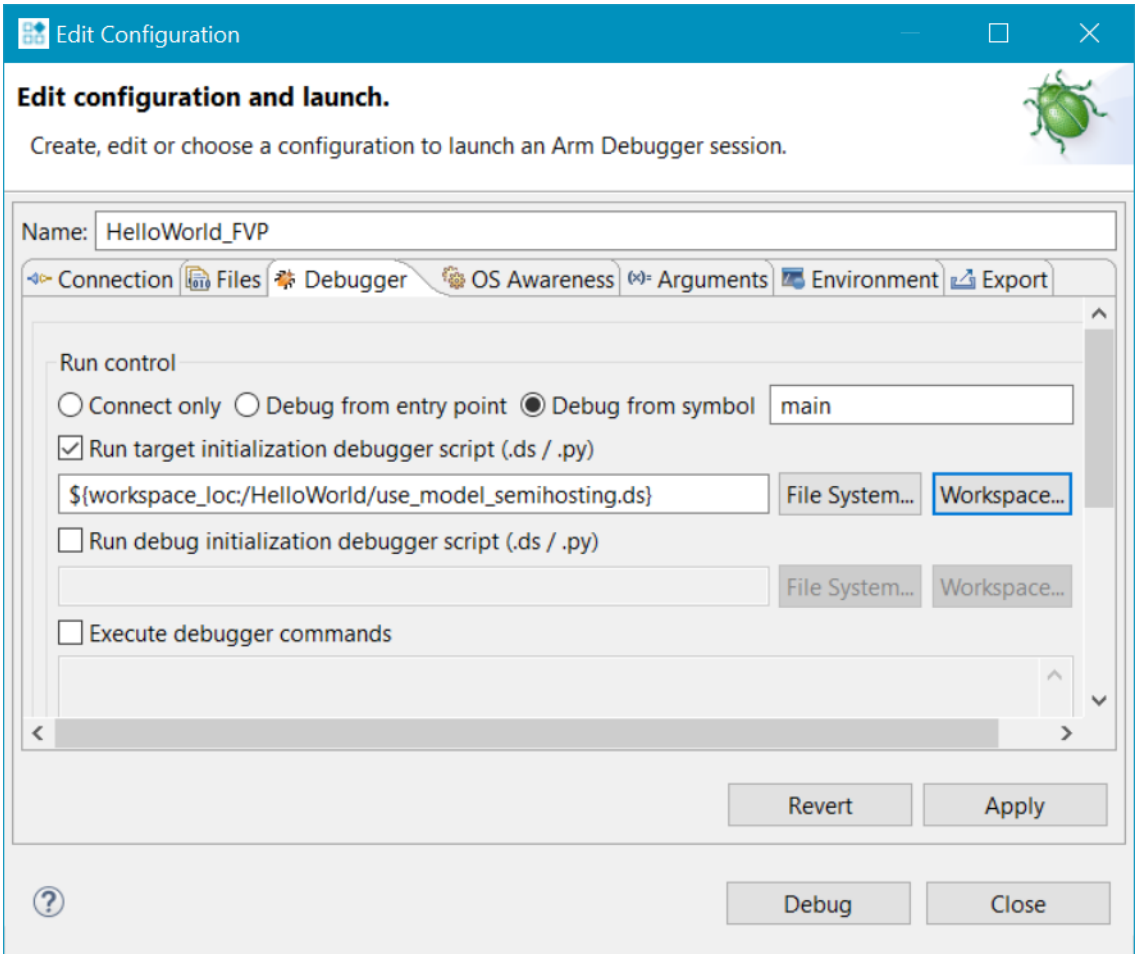


Figure 5-8 Debug from symbol main

6. 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.

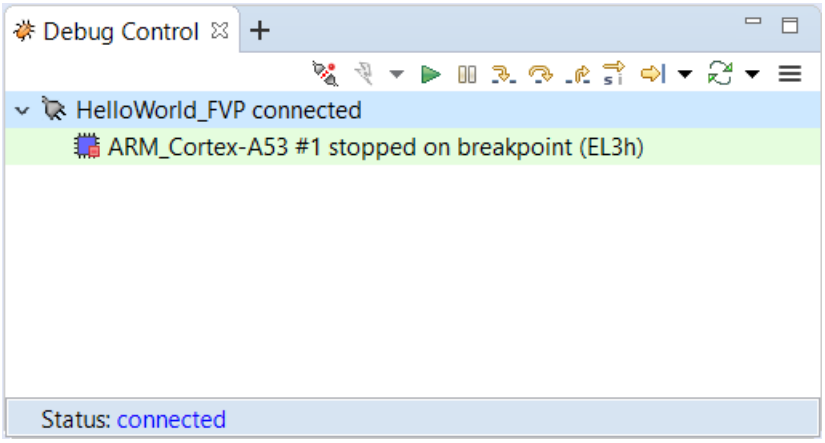


Figure 5-9 Debug Control View

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

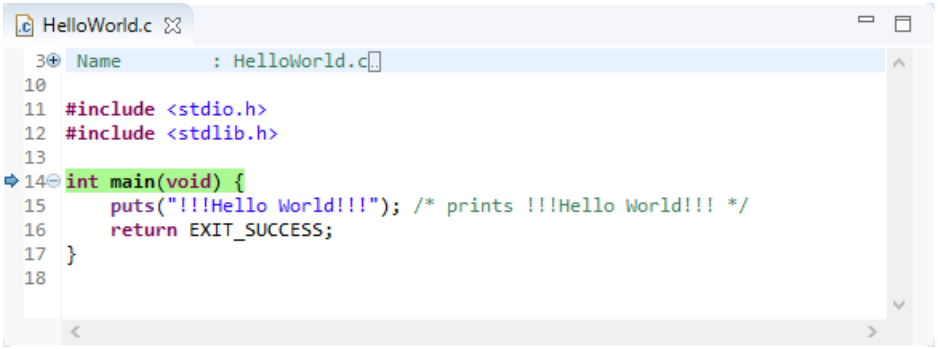


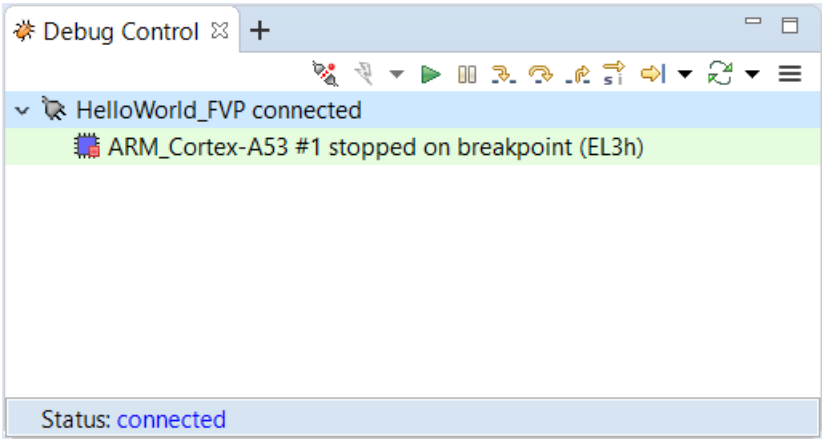
Figure 5-10 main () in code editor

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.

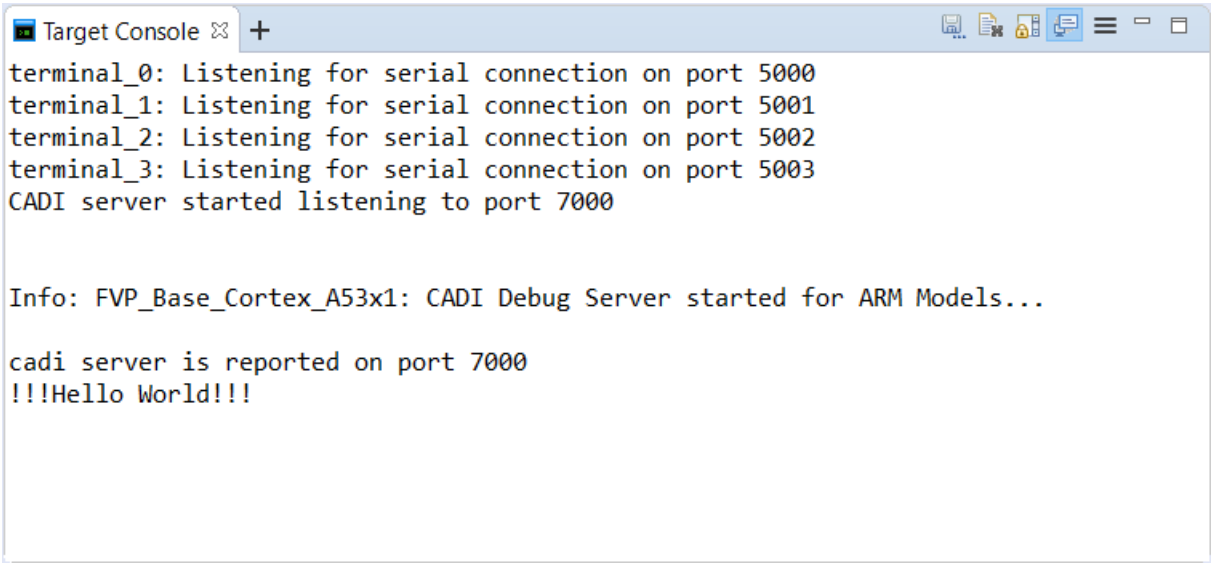


Figure 5-11 Target console output

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

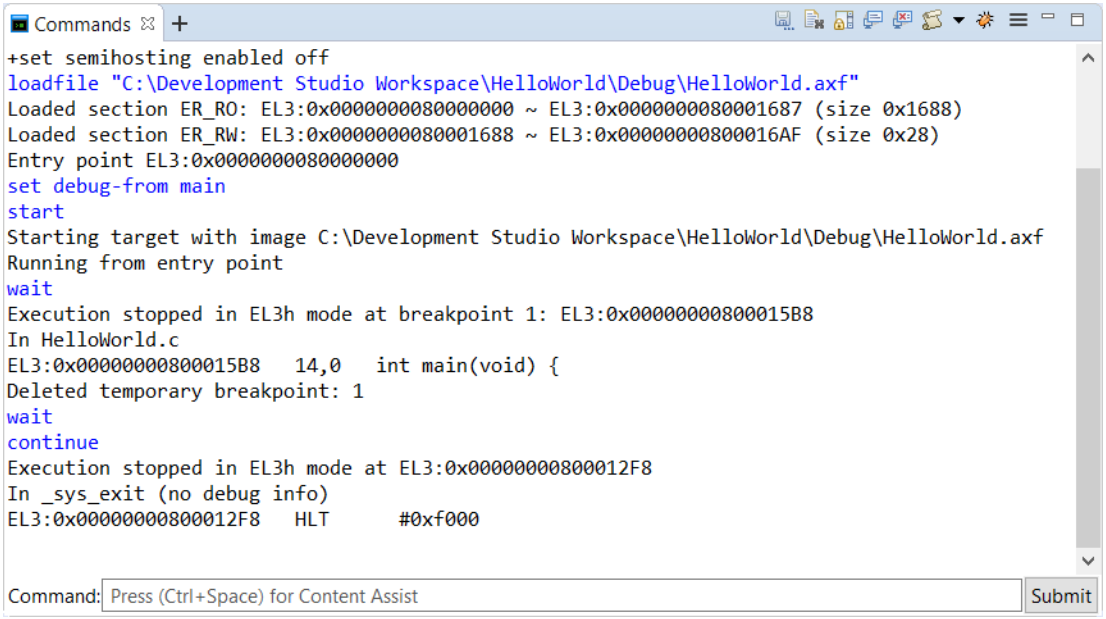


Figure 5-12 Commands view

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

HelloWorld.c | use_model_semihosting.ds

```
 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 5-13 Code Editor view

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

Disassembly

<Next Instruction> | 100

Address	Opcode	Disassembly
EL3:0x00000000800012AC	F84107FE	LDR x30,[sp],#0x10
EL3:0x00000000800012B0	D65F03C0	RET
		_sys_command_string
EL3:0x00000000800012B4	D10043FF	SUB sp,sp,#0x10
EL3:0x00000000800012B8	93407C29	SXTW x9,w1
EL3:0x00000000800012BC	AA0003E8	MOV x8,x0
EL3:0x00000000800012C0	910003E1	MOV x1,sp
EL3:0x00000000800012C4	A88127E0	STP x0,x9,[sp],#0x10
EL3:0x00000000800012C8	528002A0	MOV w0,#0x15
EL3:0x00000000800012CC	D45E0000	HLT #0xf000
EL3:0x00000000800012D0	7100001F	CMP w0,#0
EL3:0x00000000800012D4	9A9F0100	CSEL x0,x8,xzr,EQ
EL3:0x00000000800012D8	D65F03C0	RET
		_sys_exit
EL3:0x00000000800012DC	D10043FF	SUB sp,sp,#0x10
EL3:0x00000000800012E0	528004C8	MOV w8,#0x26
EL3:0x00000000800012E4	72A00048	MOVK w8,#2,LSL #16
EL3:0x00000000800012E8	93407C09	SXTW x9,w0
EL3:0x00000000800012EC	910003E1	MOV x1,sp
EL3:0x00000000800012F0	A90027E8	STP x8,x9,[sp,#0]
EL3:0x00000000800012F4	52800300	MOV w0,#0x18
EL3:0x00000000800012F8	D45E0000	HLT #0xf000
EL3:0x00000000800012FC	14000000	B _sys_exit+32 ; 0x800012FC
		__use_no_heap_region
EL3:0x0000000080001300	D65F03C0	RET
		__heap_region$guard
EL3:0x0000000080001304	D65F03C0	RET
		__Heap_ProvideMemory
EL3:0x0000000080001308	91002009	ADD x9,x0,#8
EL3:0x000000008000130C	AA0003E8	MOV x8,x0
EL3:0x0000000080001310	F9400120	LDR x0,[x9,#0]
EL3:0x0000000080001314	B4000080	CBZ x0,__Heap_ProvideMemory+28 ; 0x80001324
EL3:0x0000000080001318	EB01001F	CMP x0,x1
EL3:0x000000008000131C	91002009	ADD x9,x0,#8
EL3:0x0000000080001320	54FFFF63	B.CC __Heap_ProvideMemory+4 ; 0x8000130C
EL3:0x0000000080001324	F9400109	LDR x9,[x8,#0]
EL3:0x0000000080001328	8B090108	ADD x8,x8,x9
EL3:0x000000008000132C	EB01011F	CMP x8,x1
EL3:0x0000000080001330	540000C0	B.EQ __Heap_ProvideMemory+64 ; 0x80001348
EL3:0x0000000080001334	91001C28	ADD x8,x1,#7
EL3:0x0000000080001338	927CED08	AND x8,x8,#0xfffffffffffffff0
EL3:0x000000008000133C	8B020029	ADD x9,x1,x2
EL3:0x0000000080001340	B27D0101	ORR x1,x8,#8

Figure 5-14 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. Change the displayed number of bytes to `96` and press **Enter**.
  4. Right-click on the column headings, and select **Characters**.

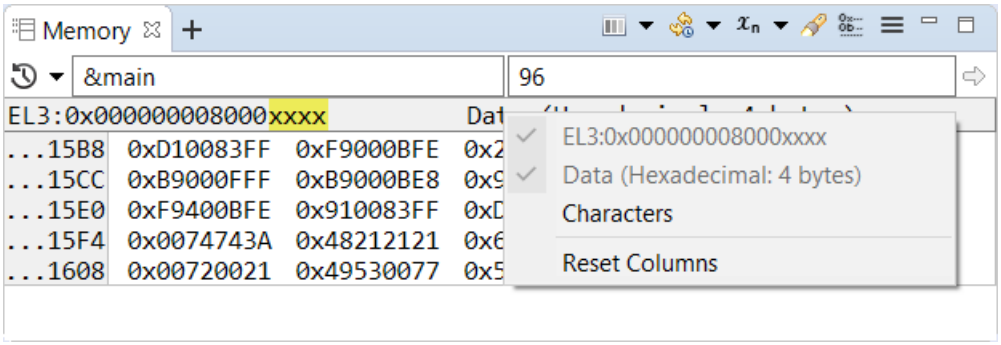


Figure 5-15 Adding Characters column to Memory view

5. Select and highlight the words *Hello World*.

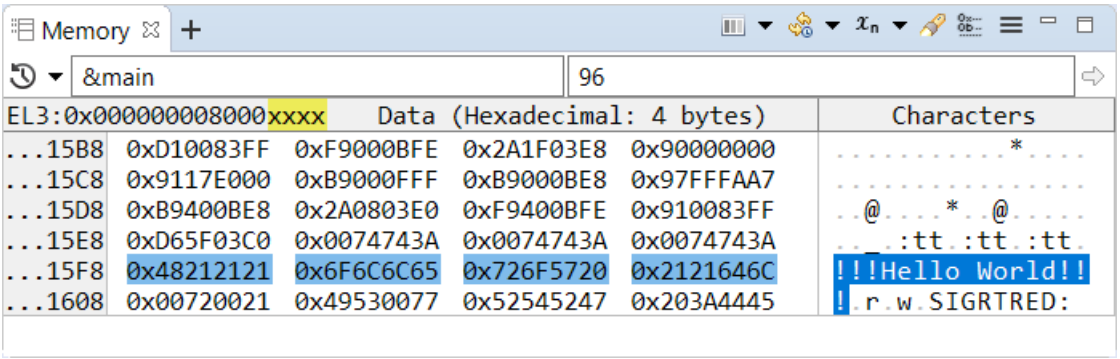


Figure 5-16 Memory view

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 5-53.

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.

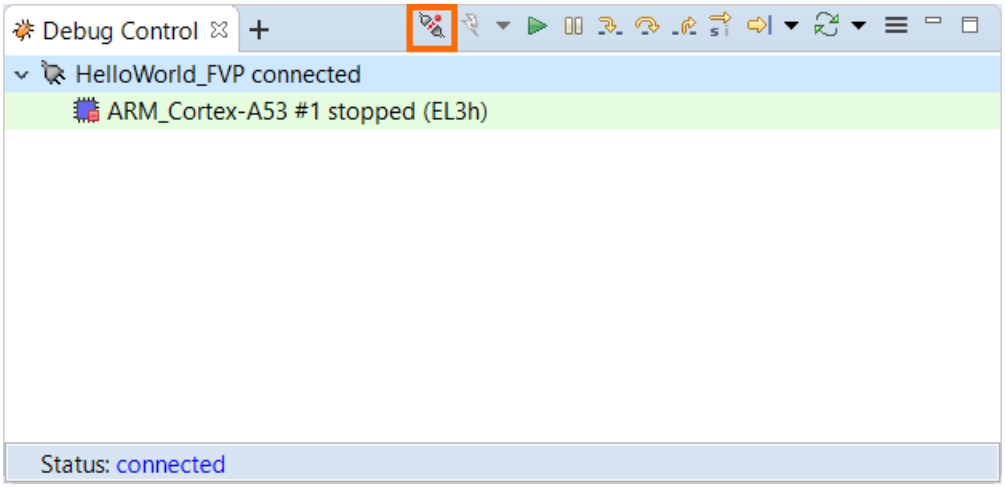


Figure 5-17 Disconnecting from a target using the Debug Control view

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

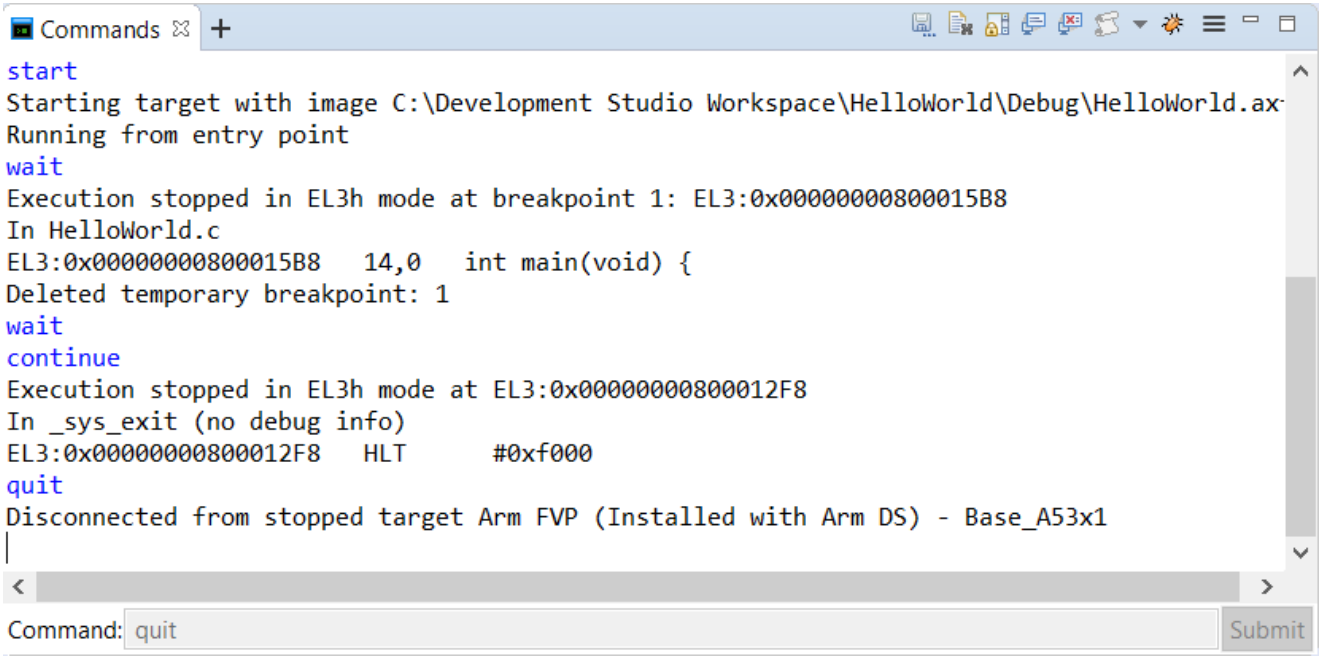


Figure 5-18 Disconnecting from a target using the Commands view

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

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.