

Kinetis Design Studio V3.0.0- User's Guide

Document Number: KDSUG
Rev. 1.0, 04/2015

Contents

Section number	Title	Page
Chapter 1		
Introduction		
1.1	System requirements.....	5
1.2	Installing Kinetis Design Studio.....	6
1.2.1	On Windows.....	6
1.2.2	On Linux.....	6
1.2.3	On Macintosh.....	7
1.3	Release Notes.....	7
1.4	Terminology.....	8
1.5	About this manual.....	9
Chapter 2		
Working with Projects		
2.1	Launching Workbench.....	11
2.2	Creating a Kinetis project.....	13
2.3	Importing an existing project.....	20
2.4	Building Projects.....	22
2.5	Debugging Projects.....	24
2.6	Flashing from file.....	31
2.7	Deleting Projects.....	31
Chapter 3		
Build Properties for Projects		
3.1	Changing Build Properties.....	34
3.2	Restoring Build Properties.....	36
3.3	Defining C/C++ Build Settings and Behavior.....	36
3.3.1	Define Build Settings.....	36
3.3.2	Define Build Behavior.....	39
3.4	Configuring Build Properties.....	42
3.4.1	Target Processor	42

Section number	Title	Page
3.4.2	Optimization.....	44
3.4.3	Warnings.....	45
3.4.4	Debugging.....	46
3.4.5	Cross ARM GNU Assembler.....	46
3.4.5.1	Cross ARM GNU Assembler > Preprocessor.....	47
3.4.5.2	Cross ARM GNU Assembler > Includes.....	47
3.4.5.3	Cross ARM GNU Assembler > Warnings.....	48
3.4.5.4	Cross ARM GNU Assembler > Miscellaneous.....	48
3.4.6	Cross ARM C Compiler.....	48
3.4.6.1	Cross ARM C Compiler > Preprocessor.....	49
3.4.6.2	Cross ARM C Compiler > Includes.....	49
3.4.6.3	Cross ARM C Compiler > Optimization.....	50
3.4.6.4	Cross ARM C Compiler > Warnings.....	50
3.4.6.5	Cross ARM C Compiler > Miscellaneous.....	50
3.4.7	Cross ARM C++ Compiler.....	51
3.4.7.1	Cross ARM C++ Compiler > Preprocessor.....	51
3.4.7.2	Cross ARM C++ Compiler > Includes.....	52
3.4.7.3	Cross ARM C++ Compiler > Optimization.....	52
3.4.7.4	Cross ARM C++ Compiler > Warnings.....	53
3.4.7.5	Cross ARM C++ Compiler > Miscellaneous.....	53
3.4.8	Cross ARM C++ Linker	54
3.4.8.1	Cross ARM C++ Linker > General.....	54
3.4.8.2	Cross ARM C++ Linker > Libraries.....	55
3.4.8.3	Cross ARM C++ Linker > Miscellaneous.....	55

Chapter 4 Appendices

4.1	Installing Kinetis SDK.....	57
4.2	Installing Kinetis SDK into KDS.....	62
4.3	Installing Drivers.....	68

Chapter 1

Introduction

The Kinetis Design Studio software development tool is a GNU/Eclipse-based development environment for Freescale Kinetis devices. It supports Cortex-M based Kinetis devices and integrates with Processor Expert and Kinetis Software Development Kit. KDS supports SEGGER J-Link/J-Trace, P&E USB Multilink Universal/USB Multilink Universal FX and CMSIS-DAP debug adapters and uses the newlib-nano C runtime library. This runtime library helps reduce the memory footprint of an embedded application.

This manual explains how to use the Kinetis Design Studio product with usage of Kinetis SDK and start effectively using Processor Expert for easier application creation and configuration with graphical user interface. This chapter presents an overview of the manual.

1.1 System requirements

Hardware	1.8 GHz processor • 2 GB of RAM •
Operating System	<ul style="list-style-type: none"> • Microsoft® Windows® 7 and Windows® 8 (all editions). Windows hosted variants of the Kinetis Design Studio software development tools are distributed as 32-bit binaries, which will run on 32-bit and 64-bit machines. • Red Hat® Enterprise Linux (RHEL), CentOS 6.4 and Ubuntu 14.04 LTS. Linux-hosted variants of the Kinetis Design Studio software development tools are distributed as 64-bit binaries, which will not work on 32-bit systems.
Disk Space	Approximately 1.5 GB of free disk space (when installing the full product)

NOTE

The Kinetis Design Studio software development tools are licensed under the terms outlined in [license.htm](#), which is found at the top of the install directory.

1.2 Installing Kinetis Design Studio

1.2.1 On Windows

The Kinetis Design Studio software development tools are installed on Windows using the Windows Installer or from the command line.

To install Kinetis Design Studio using the Windows installer:

1. Double-click the installer.
2. The Windows Installer initiates.
3. Click **Next**.
4. Follow the on-screen instructions and proceed through the installation.

To install Kinetis Design Studio from the command line, type the following to launch the Windows installer `.KDS-v3.0.0.exe /qb`. The Windows installer will launch and you can control it using the standard Window installer [command-line switches](#).

NOTE

The basic user interface will not ask any questions but will display a progress bar.

1.2.2 On Linux

To install the Kinetis Design Studio software development tools on a Linux system, use the following package files.

- `.rpm` — Use `.rpm` to install KDS software tools on systems using the RPM package manager. For example, Red Hat and CentOS.
- `.deb` — Use `.deb` to install KDS software tools on systems that use the Debian package manager. For example, Ubuntu.

installing with Red Hat package manager (RPM)

To install the Kinetis Design Studio software development tools on an Linux Standard Base (LSB)-compliant system, use the `.rpm` package file.

```
$ sudo rpm -Uvh kinetis-design-studio-3.0.0-1.x86_64.rpm
Preparing ... ##### [100%]
1: Kinetis Design Studio ##### [100%]
```

This will install the Kinetis Design Studio software development tools to the default location (`/opt/Freescale/KDS_3.0.0`).

Installing with Debian package manager (DEB).

To install the Kinetis Design Studio software development tools on Debian-like systems, including Ubuntu, use the .deb package file.

```
$ sudo dpkg -i kinetis-design-studio_3.0.0-1_amd64.deb
(Reading database ... files and directories currently installed.)
Preparing to replace kinetis-design-studio 3.0.0 (using kinetis-design-
studio_3.0.0-1_amd64 .deb) ...
Unpacking replacement kinetis-design-studio ...
Setting up kinetis-design-studio (3.0.0) ...
```

This installs the Kinetis Design Studio software development tools to the default location (/opt/Freescale/KDS_3.0.0).

NOTE

KDS includes the GCC ARM Embedded toolchain, which is built for 32 bit hosts. If you are using a 64 bit system, be sure you have the appropriate 32 bit packages installed.

- For **Ubuntu 1404** these packages are required to be installed: `libc6:i386`, `libncurses5:i386`, & `libstdc++6:i386`.
- For **RPM based packages** these packages are required to be installed: `glibc.i686` and `libncurses.so.5`.

1.2.3 On Macintosh

The Kinetis Design Studio software development tools are installed on MAC OSX using the MAC PKG installer.

To install Kinetis Design Studio using the MAC installer:

1. Double-click on the installer (PKG) file.
2. The installer initiates
3. Click **Continue**.
4. Follow the on screen instructions and proceed through the installation.

NOTE

Currently only the Segger debugger works on MAC OSX. If you are using Freedom boards, ensure that you have the Segger OpenOCD firmware installed on the board which can be found on <http://www.freescale.com/freedom> and look for your particular freedom board's getting started page.

1.3 Release Notes

Before using the Kinetis Design Studio IDE, read the release notes. These notes contain important information about last-minute changes, bug fixes, incompatible elements, or other topics that may not be included in this manual. The product comes with the release notes installed. Make sure you check the latest version of the release notes in the download section on

NOTE

The release notes for specific components of the Kinetis Design Studio IDE are located in the `Release_Notes` folder in the Kinetis Design Studio installation directory.

1.4 Terminology

The following are some of the terms used in the document.

Table 1-1. Terminology

Term	Description
Kinetis Software Development Kit (KSDK)	Software development kit that provides comprehensive software support for Freescale Kinetis devices. The KSDK includes a Hardware Abstraction Layer (HAL) for each peripheral and peripheral drivers built on top of the HAL. KSDK also contains the latest available RTOS kernels, a USB stack and other middleware to support rapid development on supported Kinetis devices.
Processor Expert	Rapid application design tool targeted for Freescale microcontrollers providing the following key features: <ul style="list-style-type: none"> • A Graphical User Interface which allows an application to be specified by the functionality needed. • An application created from Embedded Components encapsulating initialization and functionality of basic elements of embedded systems. • An automatic code generator which creates tested and optimized C code which is tuned to your application needs and the selected Freescale device. • A built-in knowledge base, which immediately flags resource conflicts and incorrect settings, so errors are caught early in design cycle allowing you to get to market faster with a higher quality product.
GNU ARM Embedded (launchpad) toolchain	Maintained by ARM, it is a GNU toolchain targeted at embedded ARM processors of the Cortex-M processor families, available from https://launchpad.net/gcc-arm-embedded . Because the GNU ARM Embedded tools are 32bit only for Linux, on Ubuntu 14.04 64-bit (and others) you may see error messages suggesting that arm-none-eabi-gcc could not be found. The tools do exist, however they system doesn't know how to run them. This is because the GCC ARM Embedded Linux tools are built for 32-bit, and compatibility packages need to be installed: See http://gnuarmeclipse.livius.net/blog/toolchain-install for details and suggested solution.

Table continues on the next page...

Table 1-1. Terminology (continued)

Term	Description
GNU ARM Eclipse plugins	Set of Eclipse plugins which integrates the GNU build tools and debugging panels into Eclipse, available from http://gnuarmeclipse.livius.net/ .
Semihosting	Originally coined by ARM, semihosting is a mechanism that enables the code running on a target to communicate and use the I/O facilities on a host computer running a debugger. The examples include: keyboard input, screen output, and disk I/O.

1.5 About this manual

Each chapter of this manual describes a different area of software development. The following table lists the contents of this manual.

Table 1-2. Manual Contents

Chapter / Appendix	Description
Introduction	This chapter.
Working with Projects	Explains how to use the Kinetis Design Studio tools to create and work with projects.
Build Properties for Projects	Explains build properties for a Kinetis project.
Installing Kinetis SDK into KDS	Lists the steps to Install Kinetis SDK into Kinetis Design Studio.
Installing Kinetis SDK	Lists the steps to install Kinetis SDK separately.
Installing Drivers	Lists the steps to install KDS debug interface device drivers on Windows and Linux.

NOTE

The processes listed in the manual are primarily for Microsoft Windows. For other supported Operating System's, the process remains the same, but some of the terms may be different based on the particular operating system. For example, the **Start** button is available in Windows.



Chapter 2

Working with Projects

This chapter explains how to use the Kinetis Design Studio to create and work with projects.

A project organizes files and various compiler, linker, and debugger settings associated with the applications or libraries you develop. You use the Kinetis New Project wizard to create new projects that group these files and settings into build and launch configurations.

2.1 Launching Workbench

To launch the Kinetis Design Studio IDE for creating, building and debugging projects:

1. Select **Start > All Programs > Kinetis Design Studio IDE**.

The **WorkSpace Launcher** dialog box appears and prompts you to select a workspace to use.

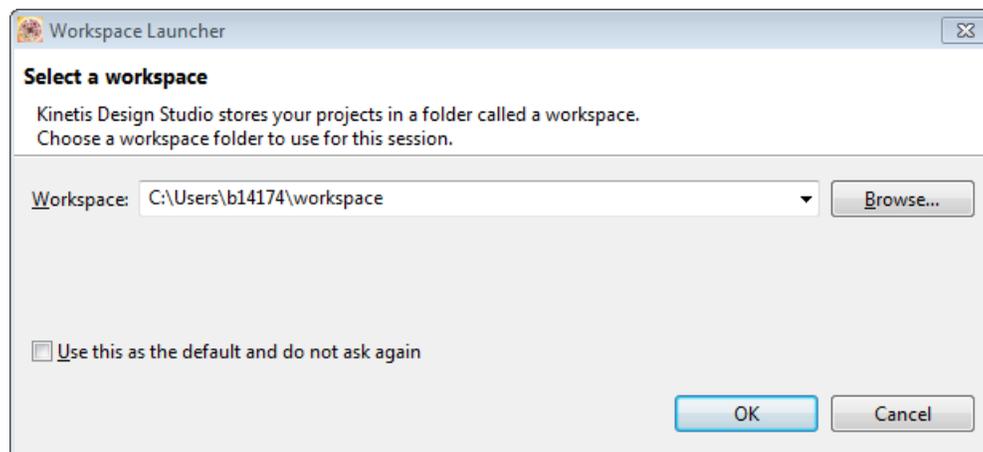


Figure 2-1. WorkSpace Launcher Dialog Box

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.
The IDE starts and displays the **Welcome** page.

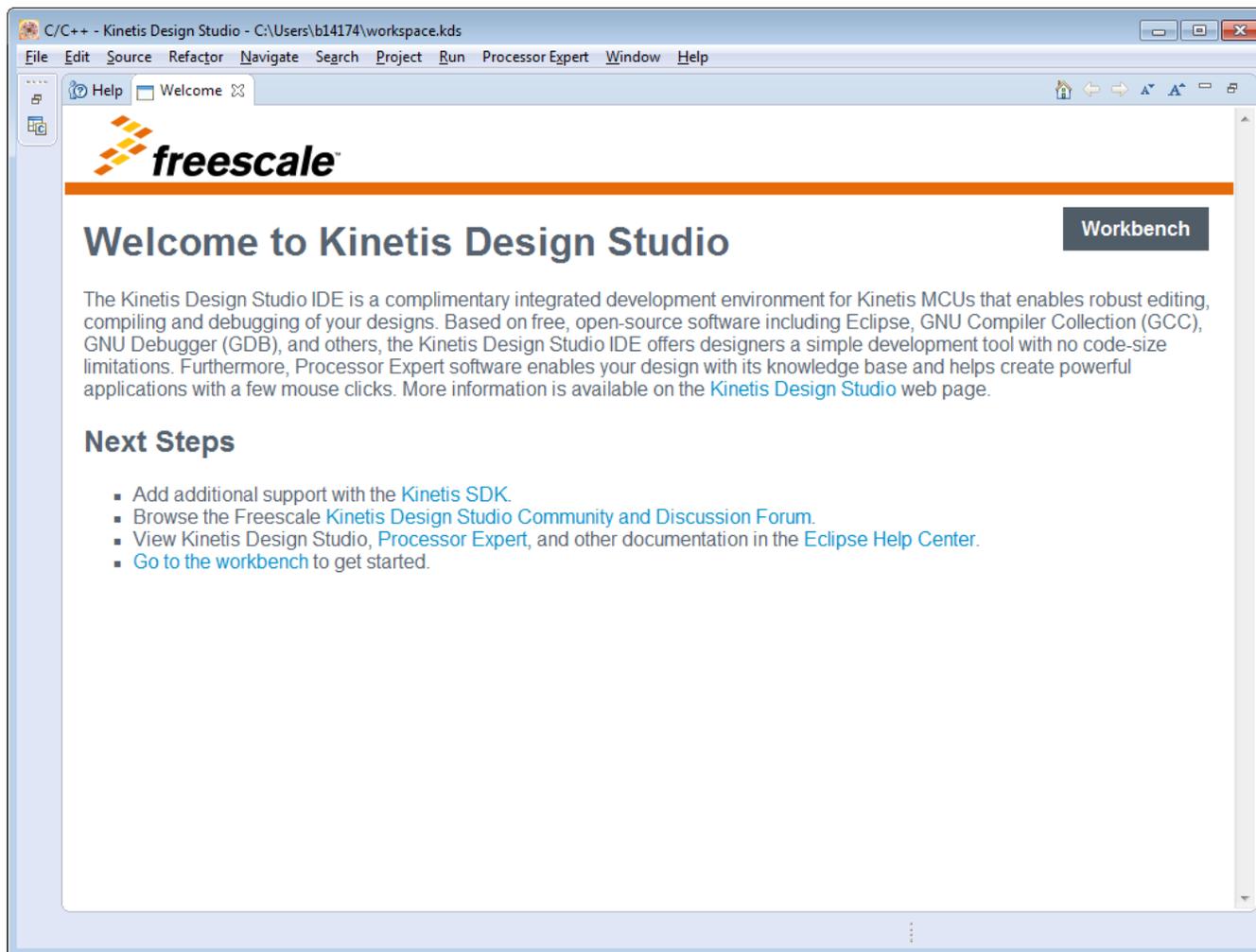


Figure 2-2. Welcome page

3. Click the **Workbench** link.
The **Workbench** window appears.

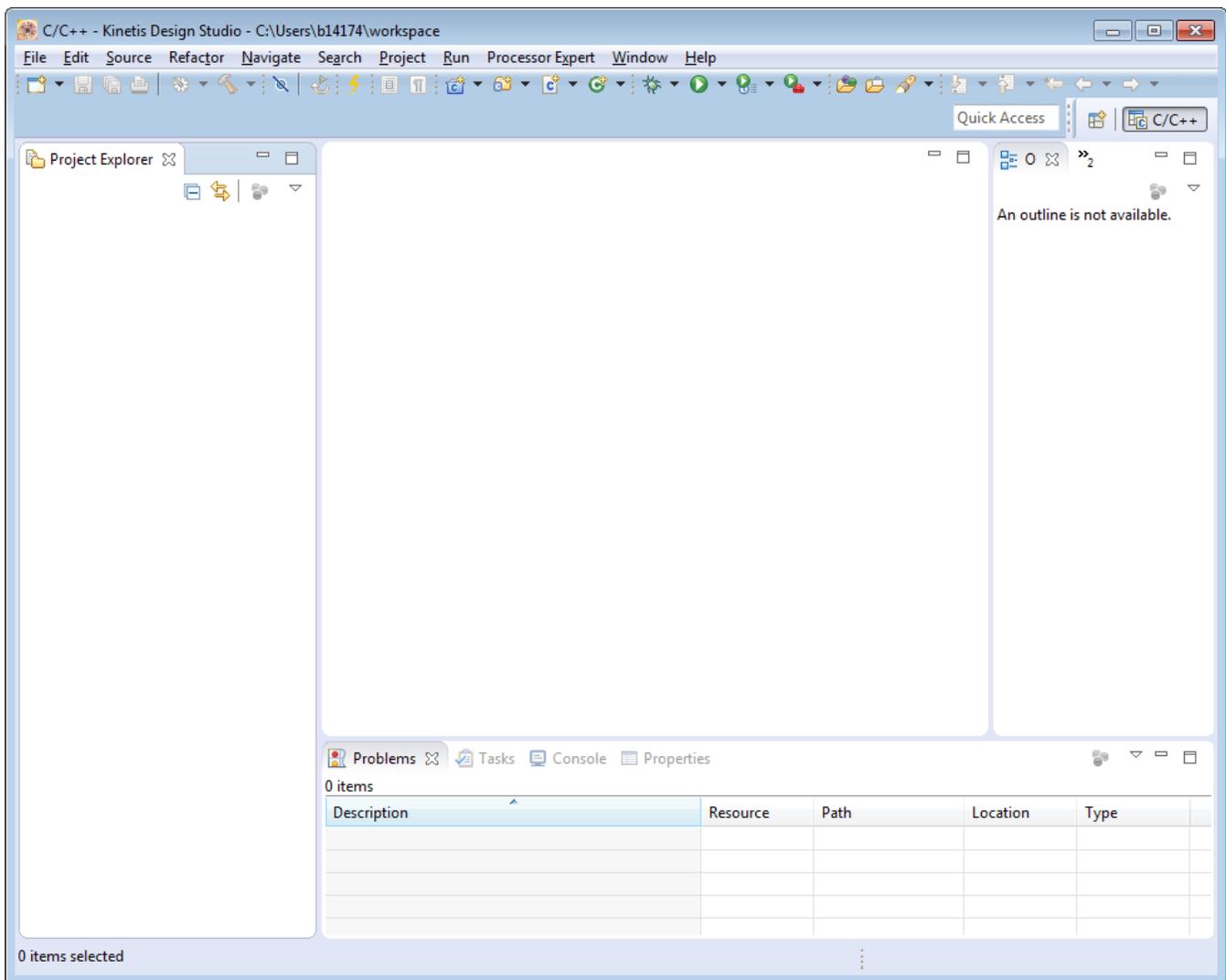


Figure 2-3. Workbench window

2.2 Creating a Kinetis project

The **Kinetis Project** wizard help you to quickly create new projects. The wizard generates a project with placeholder files and default settings (build and launch configurations) specific targets. After the project has been created, you can easily change any default setting to suit your needs.

To create a Kinetis project using the **New Kinetis Project** wizard:

1. Launch the Workbench.

NOTE

For information about launching the Workbench, refer to the topic [Launching Workbench](#).

2. Select **File > New > Kinetis Project**, from the IDE menu bar.

The **Create a Kinetis Project** page of the **New Kinetis Project** wizard appears.

3. Specify a name for the new project. For example, enter the project name as `Project1`.

NOTE

If you do not want to use the default location, clear the **Use default location** checkbox. In the **Location** text box, enter the full path of the directory in which you want to create your project including the project name. Ensure that the directory is empty. Alternatively, click **Browse** and select the desired location from the **Browse For Folder** dialog box and click **OK**. Ensure that you append the path with the name of the project to create a new location for your project.

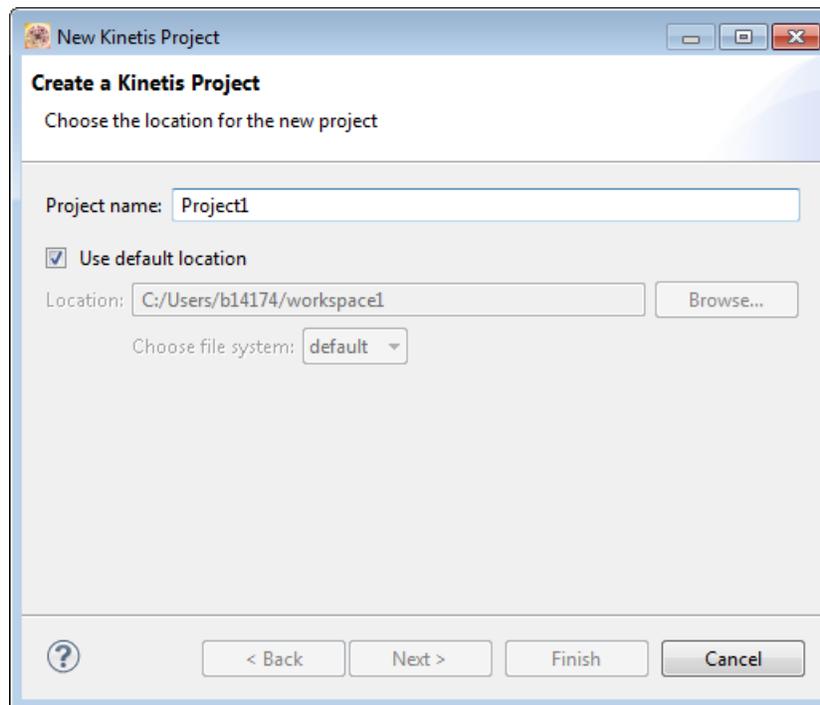


Figure 2-4. Create a Kinetis Project page

Table 2-1. Create a Kinetis Project Page Settings

Option	Description
Project Name	Enter the name for the new project in this text box. Note: Do not use the reserved/special characters/symbols such as < (less than), > (greater than), : (colon), " (double quote), / (forward slash), \ (backslash), (vertical bar or pipe), ? (question mark), @ (at), * (asterisk) in the project name. Using special characters/symbols in the project name may result in an unexpected behavior.
Use default location	Stores the files required to build the program in the Workbench's current workspace directory. The project files are stored in the default location. Clear the Use default location checkbox and click Browse to select a new location.
Location	Specifies the directory that contains the project files. Click Browse to navigate to the desired directory. This option is available only when Use default location checkbox is clear.

4. Click **Next**.

The **Devices** page appears.

5. Expand the desired tree control and select the derivative or board you would like to use. For example, select Processors > Kinetis K > MK60 > MK64F (120 MHz) > MK64FN1M0xxx12.

NOTE

You can write the part of the derivative name in the filter column and the selection is filtered. For example, type K64F and select project board support or derivative project or clean project for the specific derivative.

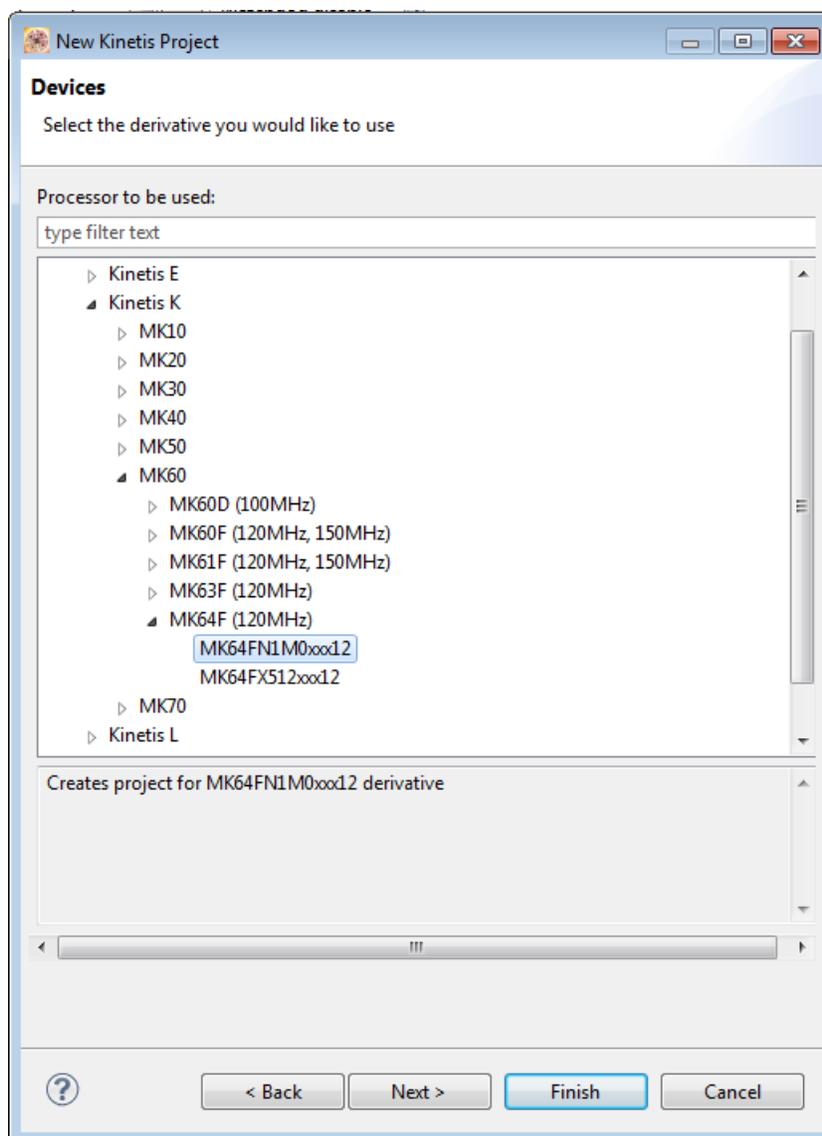


Figure 2-5. Devices page

NOTE

You can click **Finish** at any step in the project wizard to save the project with the default settings. However, it is recommended that you click **Next** to ensure that the project settings match your needs.

6. Click **Next**.

The **Rapid Application Development** page appears. This page helps you to configure use Processor Expert for configuration and KSDK. Processor Expert is included with Kinetis Design Studio software, but KSDK must be installed separately.

See [Installing Kinetis SDK into KDS](#) and [Installing Kinetis SDK](#) for details.

NOTE

When the Kinetis SDK is installed, the default selection uses the Kinetis SDK. By default, Processor Expert is not selected.

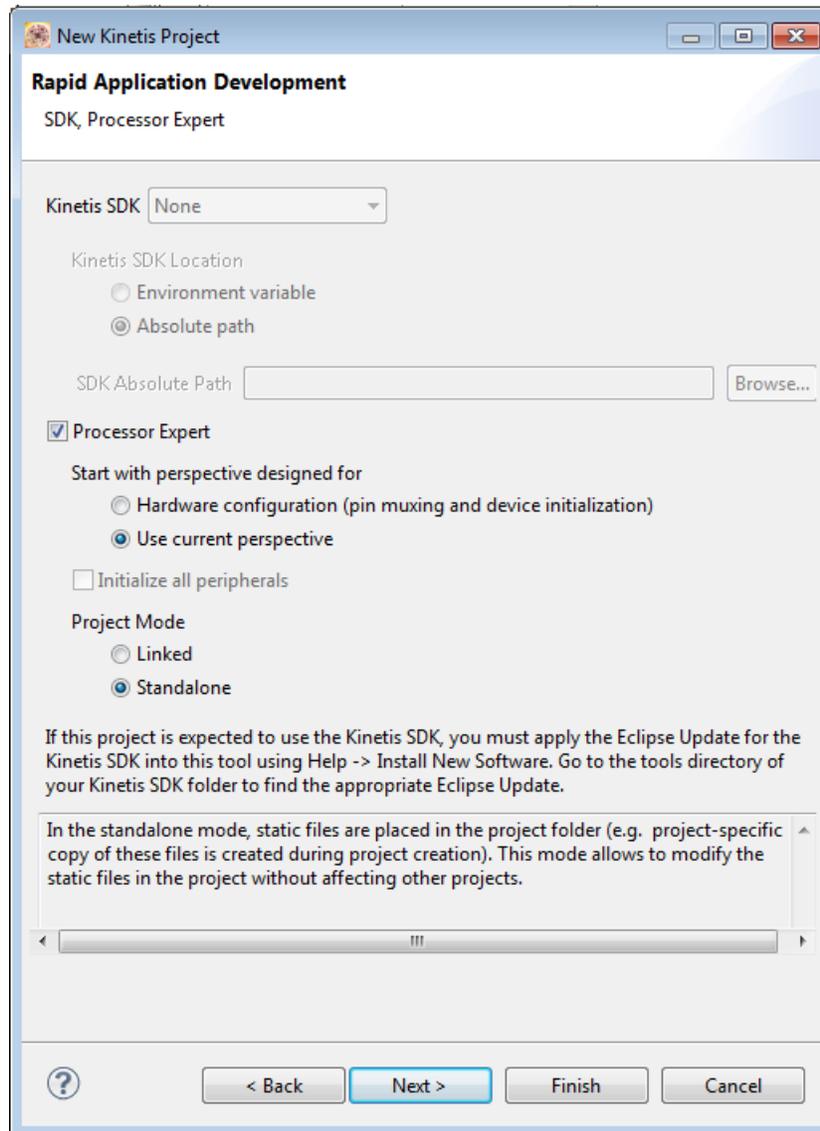


Figure 2-6. Rapid Application Development page

7. The default Kinetis SDK option is set to None. To use the Kinetis SDK, you must apply the Eclipse Update for the Kinetis SDK into this tool, and then select the Kinetis SDK library version you want to use. You can use Environmental variable <KSDK_PATH> or select the absolute path to the KSDK installation folder.

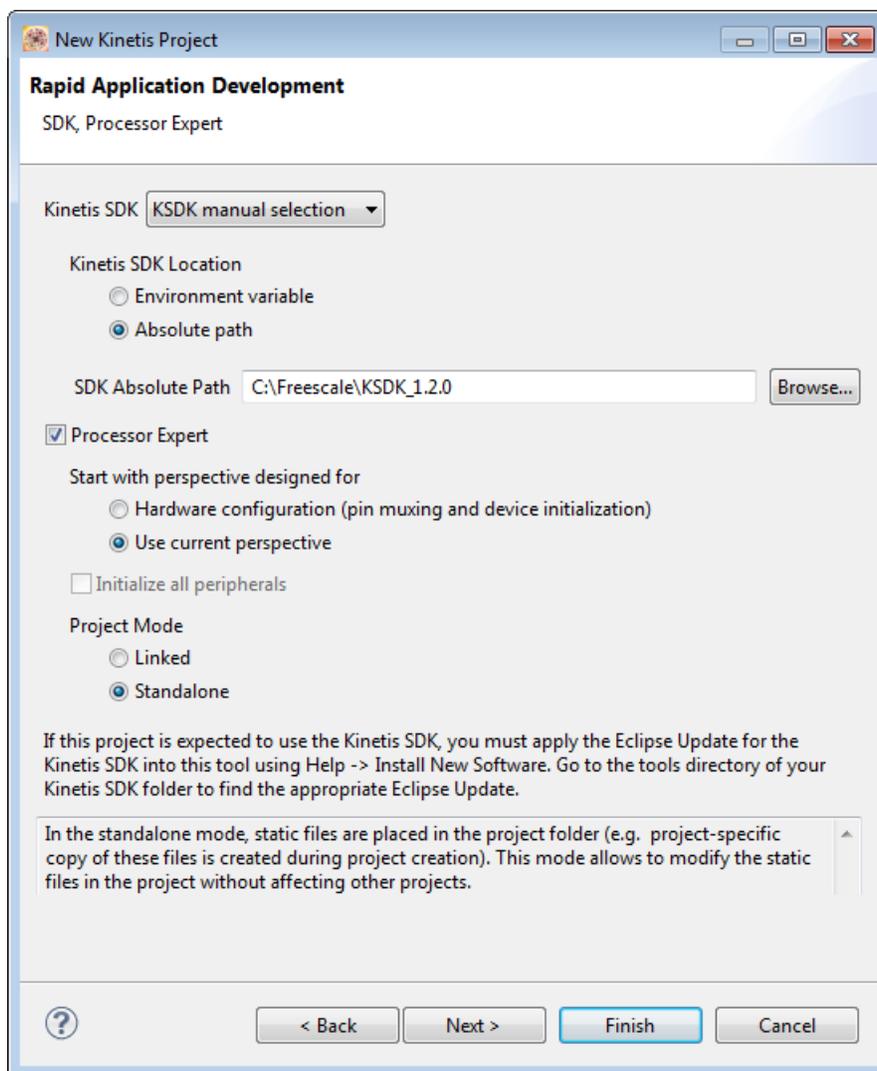


Figure 2-7. Rapid Application Development page

NOTE

See the Getting Started with SDK guide for more information on how to apply Kinetis SDK into Kinetis Design Studio.

8. Check the Processor Expert checkbox.
9. Choose the perspective you want to start with.
 - **Hardware configuration (pin muxing and device initialization)** - Select to generate the pin muxing and device initialization code, including low-level drivers.
 - **Use current perspective** - Select to keep the default perspective settings.
10. Select the appropriate **Project Mode**.

- **Linked** - The static files are linked from corresponding repositories into a project. In this mode, the repositories may be shared in other projects. Modification of these files affect other projects where the static files are linked.
 - **Standalone** - The static files are placed in the project folder. This means a copy of these files is created for a specific project during project creation. In this mode, you can modify the static files without affecting other projects. Standalone mode makes it easier to share projects among teams.
11. Click **Next**. The Target compiler page appears.

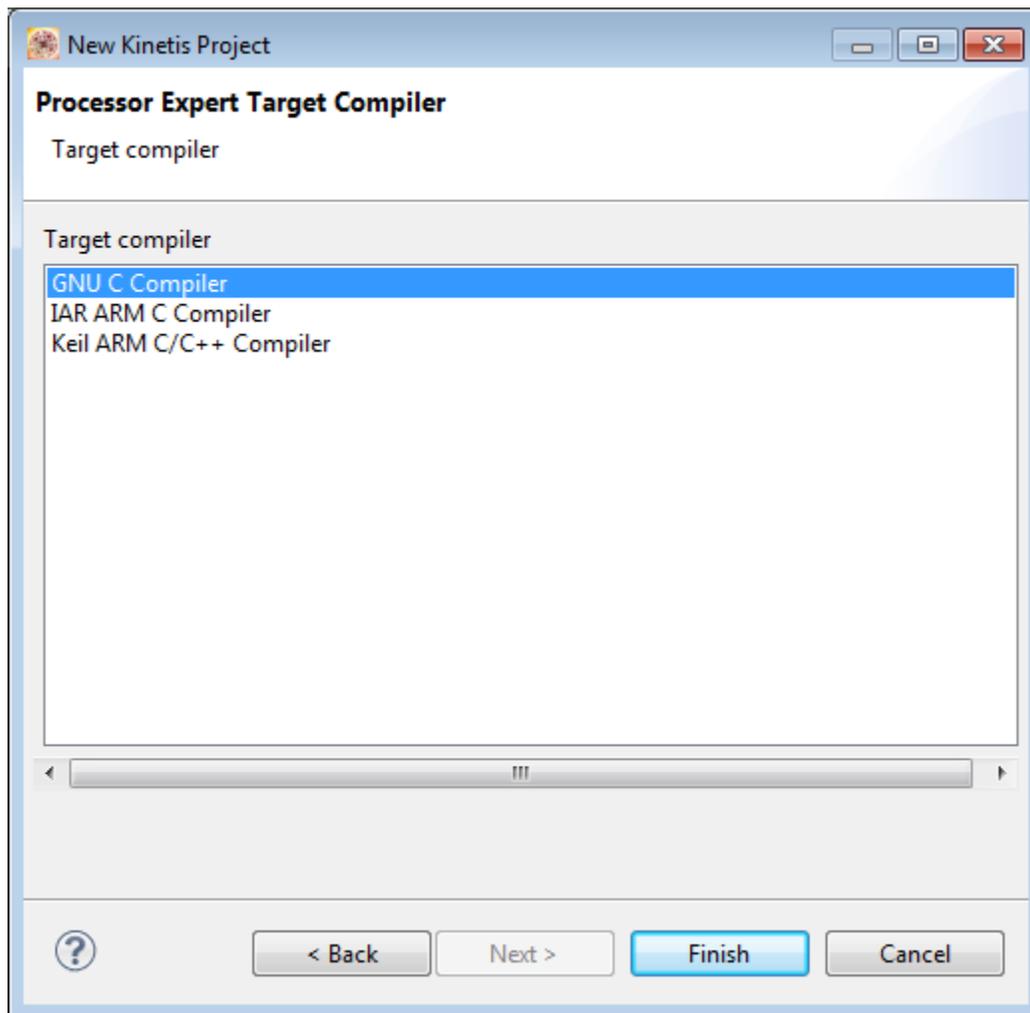


Figure 2-8. Processor Expert Target Compiler

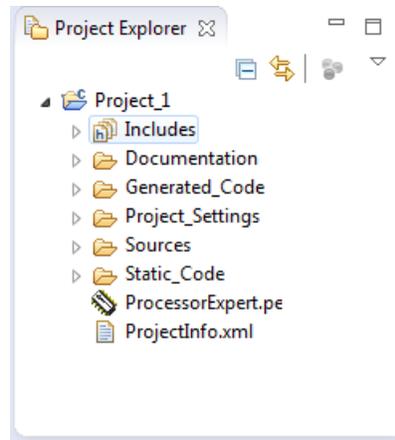
12. Select the required compiler. Select the GNU C Compiler option, if you want KDS to build code.

NOTE

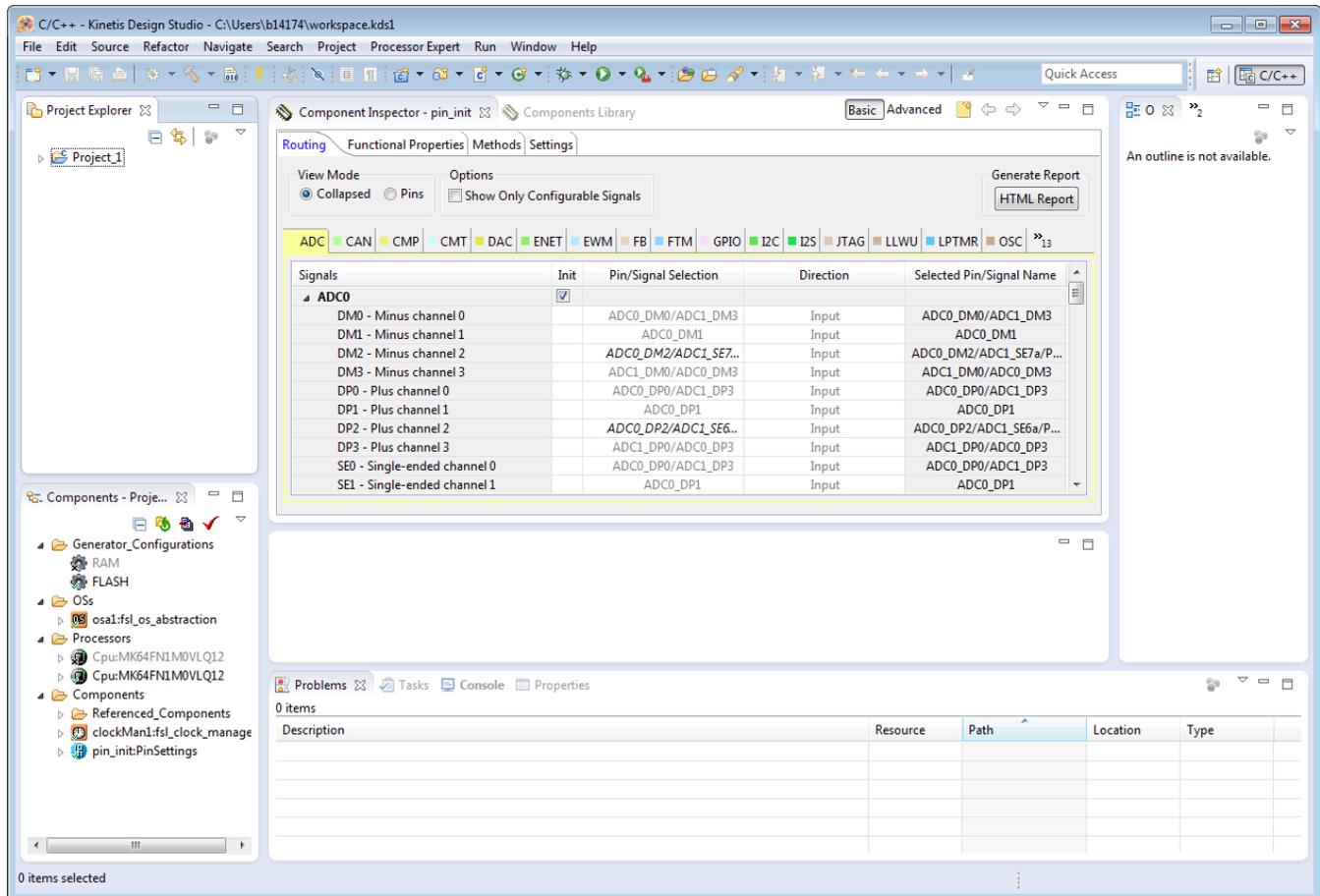
The KDS contain GNU C Compiler installed by default. Other compiler must be installed in KDS by users.

13. Click **Finish**.

Importing an existing project



The new project is ready for use. You can now customize it by adding your own source code files, changing debugger settings, or adding libraries. To create a new source file under the project, right click on the project and select New > Source File. Alternatively you can drag and drop existing source files, header files, directories into the project.



2.3 Importing an existing project

This section should explain how to import an existing Kinetis SDK project. Explain that the Kinetis SDK needs to be installed first for this, then go through the steps to import one of the Kinetis SDK projects.

To import an existing Processor Expert project:

1. Select **File > Import**, from the IDE menu.

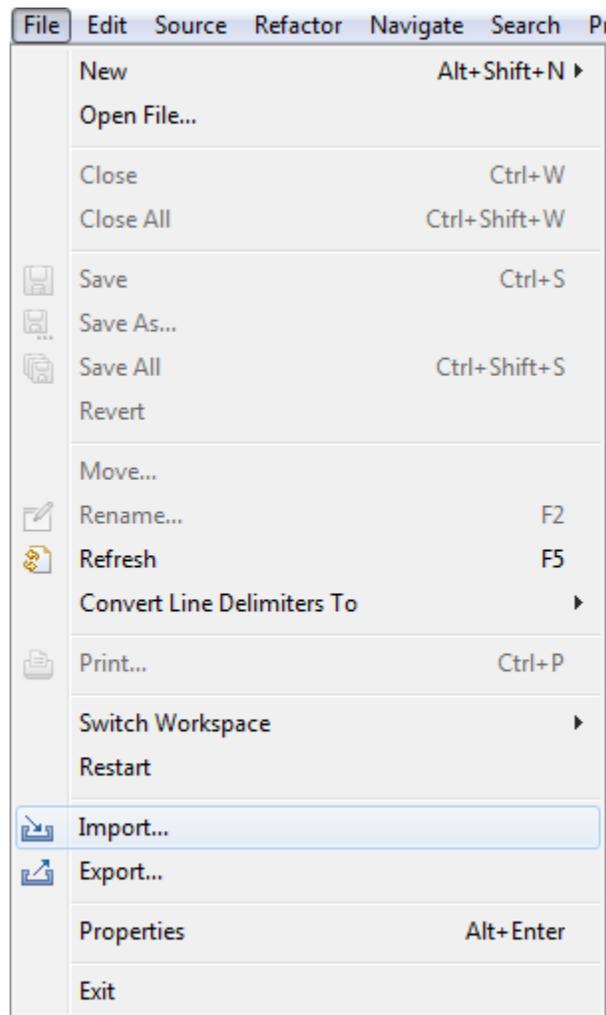


Figure 2-9. Select import option

The import dialog appears.

2. Expand the **General** tree and select **Existing Projects into Workspace**.

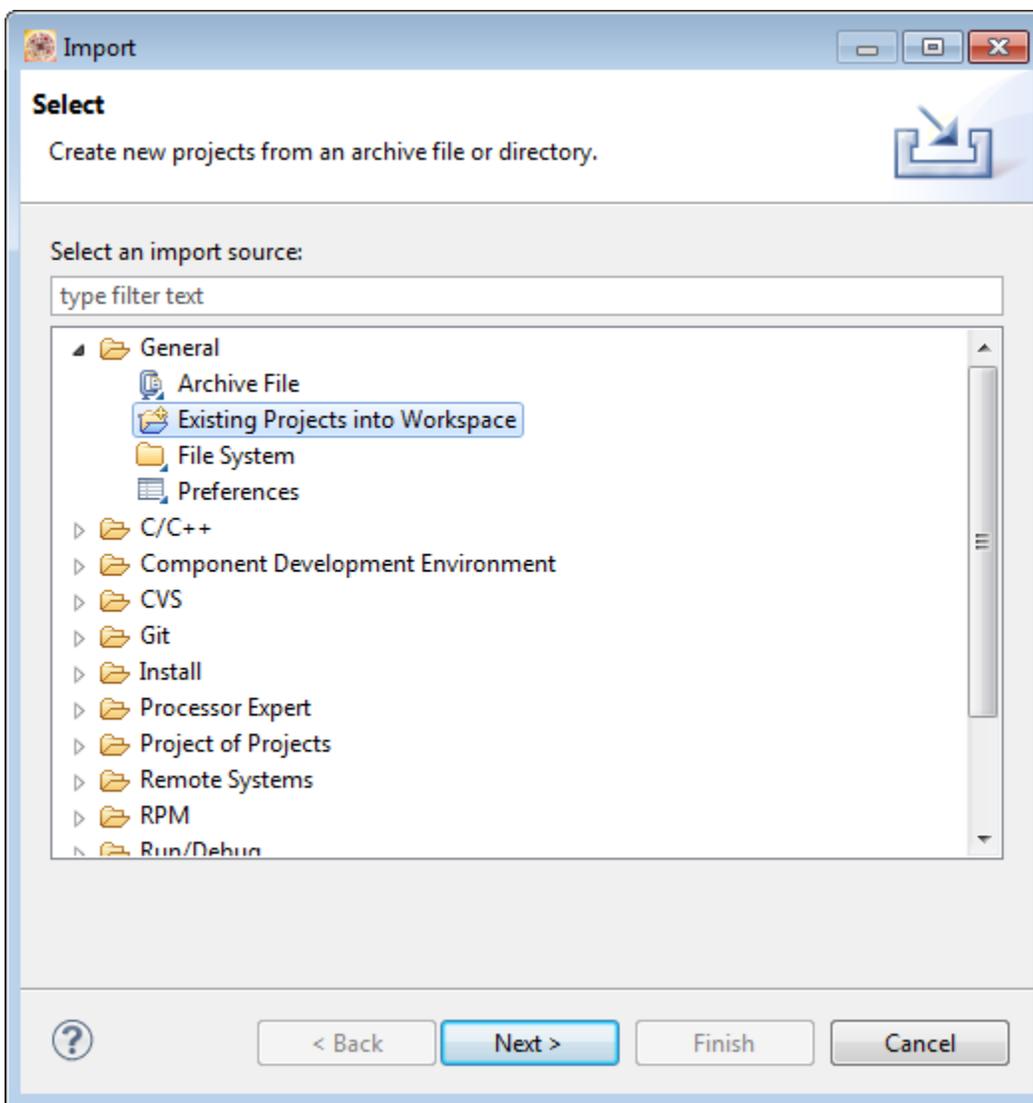


Figure 2-10. Select an import source

3. Click **Next**. The Import projects screen appears.
4. Click **Browse** and select the root directory to search for an existing Eclipse project.
5. Select the projects you want to import in your Workspace.
6. Click **Finish**.

The imported project appears in the **Project Explorer** view.

2.4 Building Projects

The recently built Kinetis Design Studio project is pre-configured and you can easily build the project for your Freescale Kinetis MCU based target board. However, if you want to change the configuration of the project you can adjust the build properties. For more information on build properties, see [Build Properties for Projects](#).

NOTE

In large workspaces, building the entire workspace can take a long time if you make changes with a significant impact on dependent projects. Often there are only a few projects that really matter to you at a given time.

To build only the selected projects, and any prerequisite projects that need to be built in order to correctly build the selected projects, click  or right-click and select **Project > Build Project** from the Kinetis Design Studio IDE menu bar.

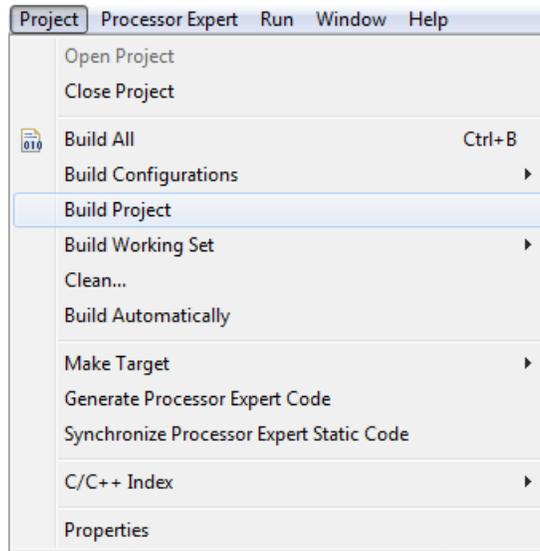


Figure 2-11. Project Menu - Build Project

Alternatively, select **Project > Build All** .

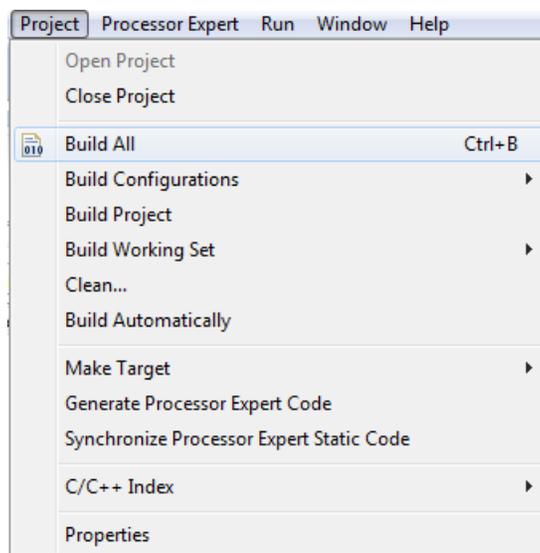


Figure 2-12. Project Menu - Build All

Alternatively, you can right-click on a selected project and select **Build Project**.

2.5 Debugging Projects

When you use the **Kinetis New Project** wizard to create a new project, the wizard sets the debugger settings of the project's launch configurations to default values. You can change these default values based on your requirements.

To debug a project, perform these steps.

1. Launch the IDE.
2. Click  The **Launch Configuration Selection** dialog appears.

Alternatively, you can select **Run > Debug Configurations** from the IDE menu bar. If you choose to select **Run > Debug Configurations**, proceed to Step 4.

NOTE

The Kinetis Design Studio software development tools supports the following debug adapters for debugging applications on a Freescale Kinetis device.

- OpenOCD - On-board OpenSDA debug interface running the ARM® mbed™ project CMSIS-DAP firmware.
- P&E USB Multilink Universal and USB Multilink Universal FX debug adapters.
- SEGGER J-Link and J-Trace debug adapters.

NOTE

When using either the OpenOCD, SEGGER J-Link or P&E Multilink debug interface, the relevant device drivers need to have been installed. For information on installing drivers, see [Installing Drivers](#).

NOTE

The launch/debug configurations are populated with the default settings: best if you go into the launch configuration/settings to verify the correct USB port is selected/etc.

3. Select the launch configuration you want to debug.

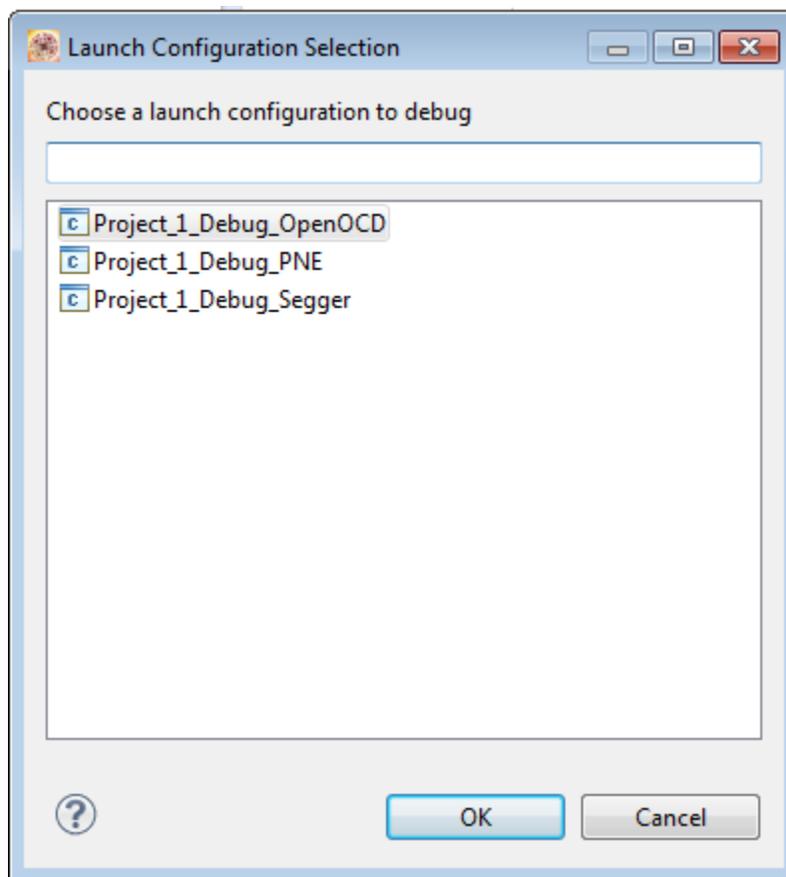


Figure 2-13. Select launch configuration

4. Click **OK**.

The IDE uses the settings in the launch configuration to generate debugging information and initiate communications with the target board. The **Debug Configurations** dialog appears. The left side of this dialog box has a list of debug configurations that apply to the current application.

5. Expand the tree and select the debug configuration that you want to modify.

The figure below displays the **Debug Configurations** dialog box with the settings for the debug configuration you selected.

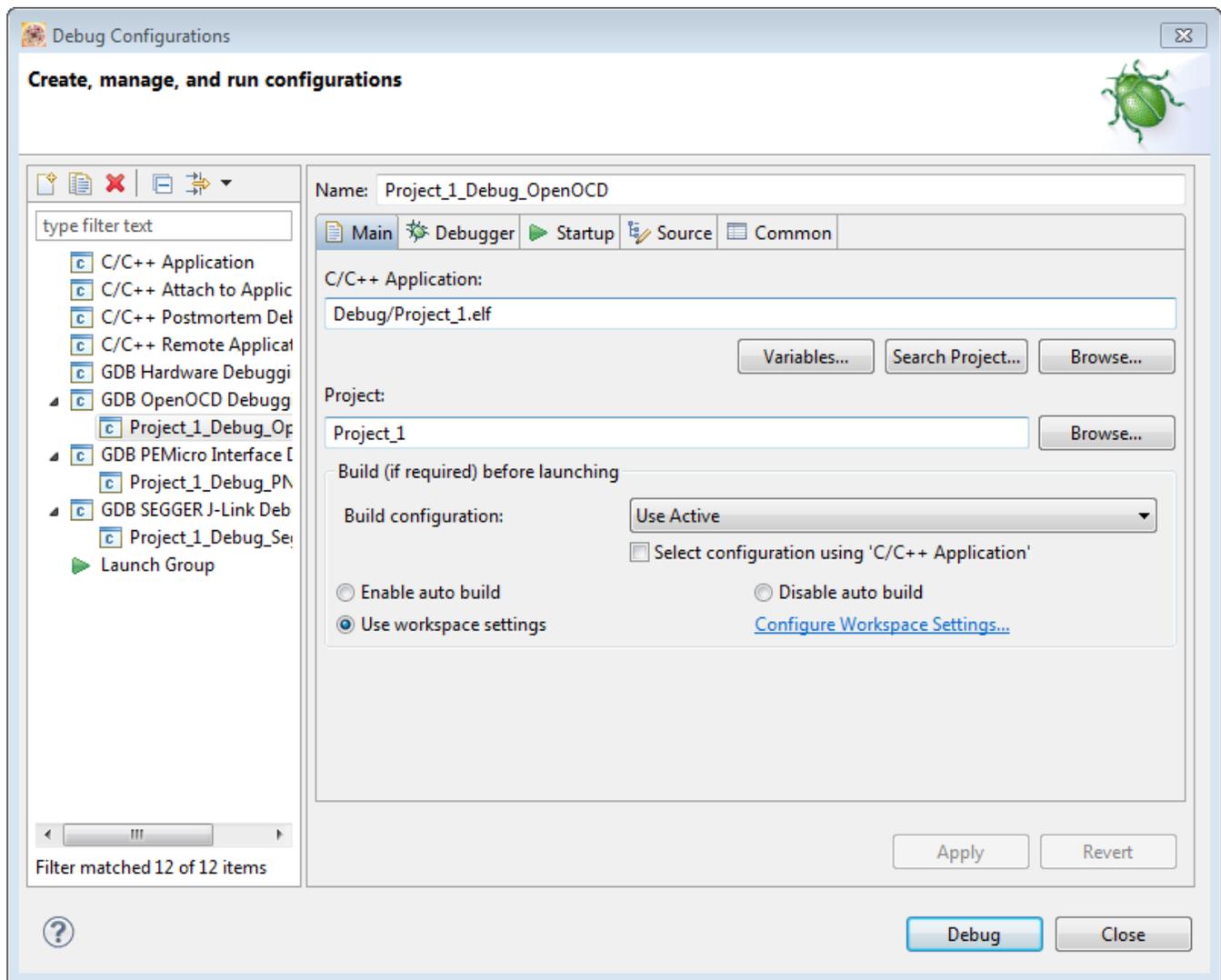


Figure 2-14. Debug configuration

6. In the **Main** tab, ensure that the correct Project and C/C++ Application is selected.
7. Select the **Debugger** tab.

The **Debugger** page appears in the area beneath the tabs.

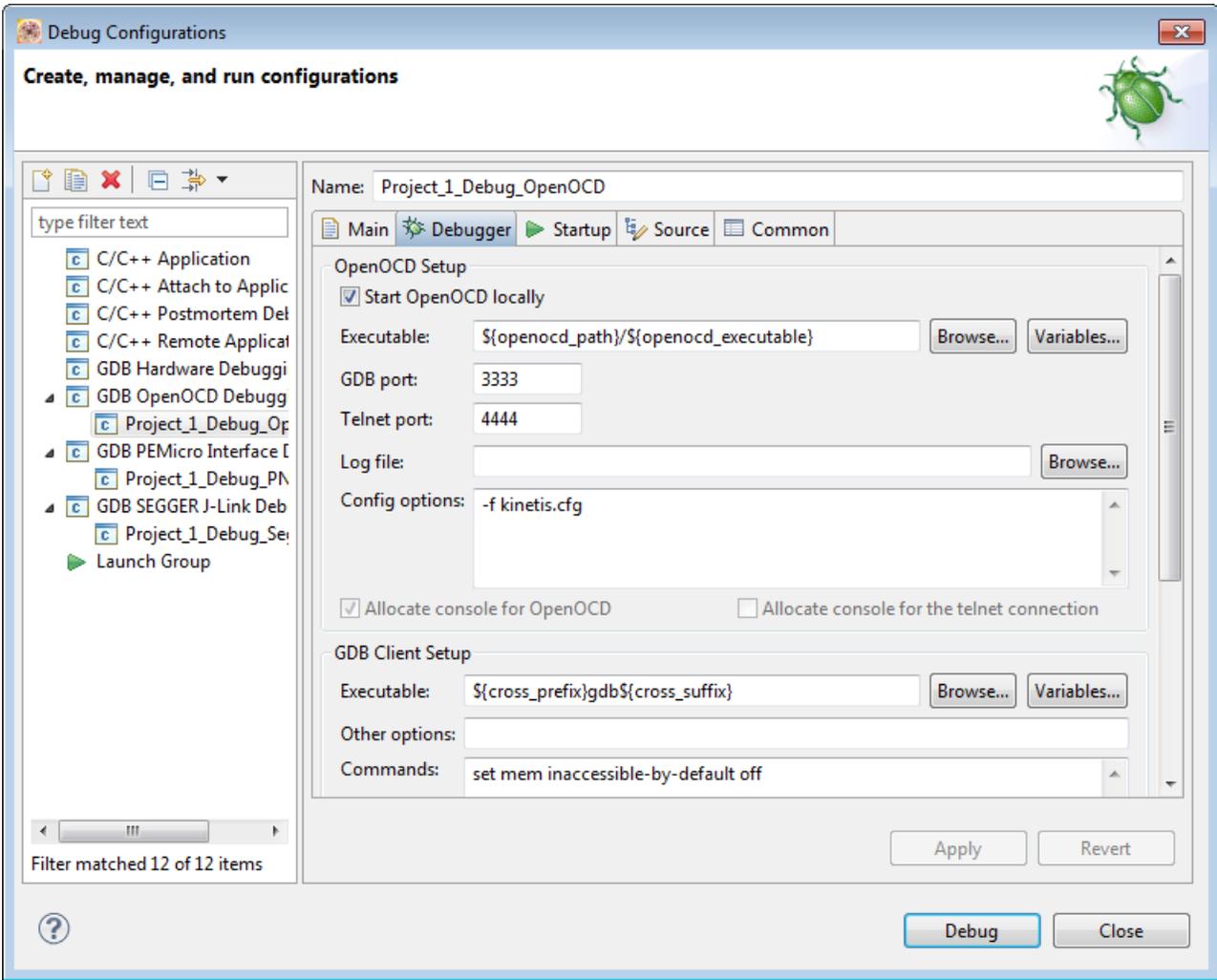


Figure 2-15. Debug Configurations Dialog Box - Debugger Page - OpenOCD

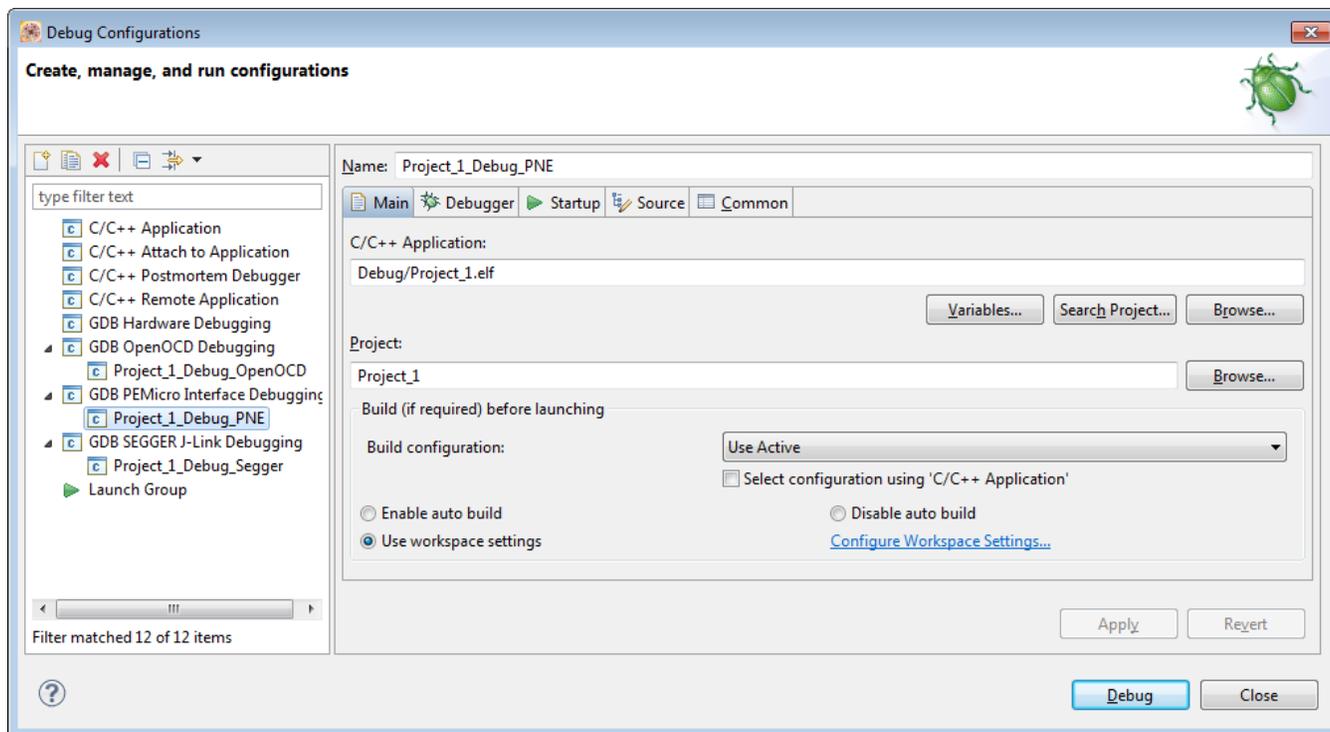


Figure 2-16. Debug Configurations Dialog Box - Debugger Page - PnE

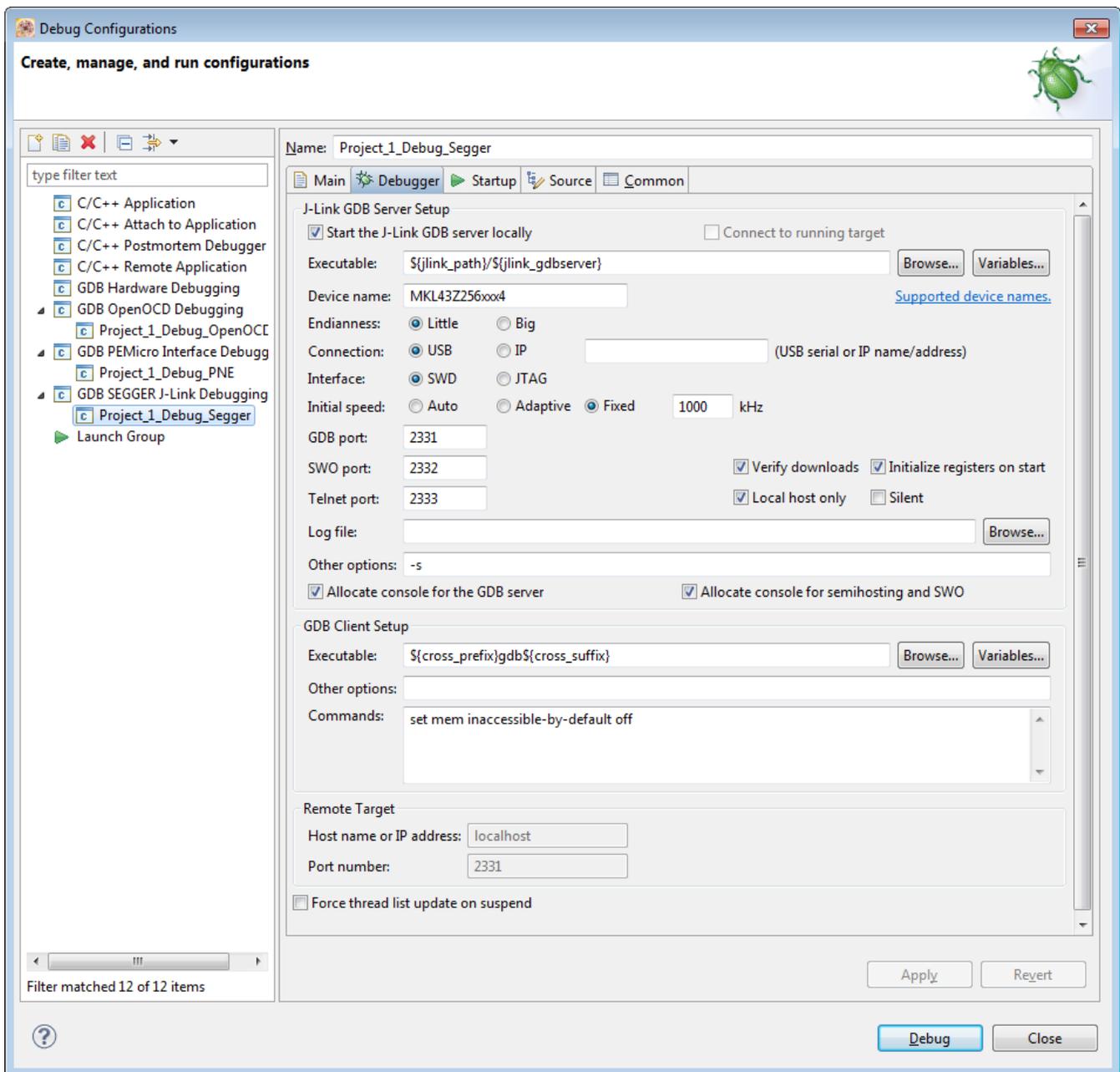


Figure 2-17. Debug Configurations Dialog Box - Debugger Page - Segger

8. Based on the debug interface you selected, change the appropriate debugger settings. The below mention changes will connect to the remote target as a `localhost`.

Table 2-2. Debug interface settings

Debug interface	Debugger Settings
OpenOCD	Set the value of Other options under GDB Client Setup group to <code>-f kinetis.cfg</code> .
P&E	Select the Device name for your Freescale Kinetis device from the dropdown list.

Table continues on the next page...

Table 2-2. Debug interface settings (continued)

Debug interface	Debugger Settings
	<p>NOTE: If you are using the P&E OpenSDA firmware, select the Interface as OpenSDA Embedded Debug - USB Port.</p>
SEGGER J_Link	<p>Enter the Device name for your Freescale Kinetis device. You can use the link Supported device names to help you with your selection.</p> <p>NOTE: SEGGER software enables protection from accidental permanently locking of devices by providing two variants of each Freescale Kinetis device. The default option does not allow mass erase, while the alternate labeled as allow security enable mass erase. Thus, you should not use allow security devices without a good reason.</p>

9. Select the **Startup** tab.
10. Based on the debug interface you selected, change the appropriate startup settings.

Table 2-3. Startup settings

Debug interface	Startup Settings
OpenOCD	No changes required.
P&E	No changes required.
SEGGER J_Link	<p>Clear the Enable SWO checkbox.</p> <p>NOTE: if the board/device has SWO wired to the debug port, then the SWO can remain enabled.</p>

11. Click **Apply** to save the new settings.
12. Click **Debug** to start the debugging session.

NOTE

If the Debug Perspective is not open, you will be prompted to open the Debug Perspective.

13. Select Yes to switch perspective. The Debug Perspective will open and the embedded application will break on the breakpoint set on main.

You just finished starting a debugging session and attaching the debugger to a process.

NOTE

You can click the **Revert** button in the Debug Configurations dialog to undo any of the unsaved changes. The IDE restores the last set of saved settings to all pages of the **Debug Configurations** dialog box. Also, the IDE disables **Revert** until you make new pending changes.

2.6 Flashing from file

Kinetis Design Studio includes a functionality to program or flash a device without having a project. This is called 'Flash from file'. With this it will setup a special debug configuration which is used to specify the binary. For example: ELF file to be downloaded.

To flash a project in Kinetis Design Studio:

1. Launch the IDE.
2. Click the  icon on the IDE menu bar. The **Debug Configuration** dialog box appears.
3. In the **Main** tab, ensure that the correct Project and C/C++ Application is selected.
4. Select the **Debugger** tab.
5. Configure as discussed in step 7-10 of [Debugging Projects](#).
6. Click **Flash**. The embedded application is flashed to your Freescale Kinetis device. Once completed, power cycle the board. The embedded application should boot and run.

2.7 Deleting Projects

To delete a project, follow these steps.

1. Select the project you want to delete in the **Projects Explorer** view.
2. Select **Edit > Delete**.

The **Delete Resources** dialog box appears.

NOTE

Alternatively, you can also select **Delete** from the context menu when you right-click on the project.

3. Check the **Delete project contents on disk (cannot be undone)** checkbox if you want to delete the contents of the selected project. Else, clear the **Delete project contents on disk (cannot be undone)** checkbox.

NOTE

You will not be able to restore your project using *Undo*, if you select the **Delete project contents on disk (cannot be undone)** option.

4. Click **OK**.

The project is removed from the **Project Explorer** view.

Chapter 3

Build Properties for Projects

This chapter explains build properties for a Kinetis project. The Kinetis New Project wizard uses the information it gathers from you to set up the project's build and launch configurations.

A project's build configuration contains information on the tool settings used to make the program. For example, it describes the compiler and linker settings, and the files involved, such as source and libraries.

A project's launch configuration describes how the IDE starts the program, such as whether it executes by itself on a target, or under debugger control. Launch configurations also specify the core the program executes on (if the target processor has multiple cores). They also specify the connection interface and communications protocol that the debugger uses to control the environment that the program executes in.

NOTE

The settings of the Kinetis Design Studio IDE's build and launch configuration correspond to an object called a target made by the classic Kinetis Design Studio IDE.

When the new project wizard completes its process, it generates launch configurations with names that follow the pattern `projectname - configtype - targettype`, where:

- `projectname` represents the name of the project
- `configtype` represents the project's name, which usually describes the build configuration
- `targettype` represents the type of target software or hardware on which the launch configuration acts

For each launch configuration, you can specify build properties, such as:

- additional libraries to use for building code
- behavior of the compilers, linkers, assemblers, and other build-related tools
- specific build properties, such as the byte ordering of the generated code

3.1 Changing Build Properties

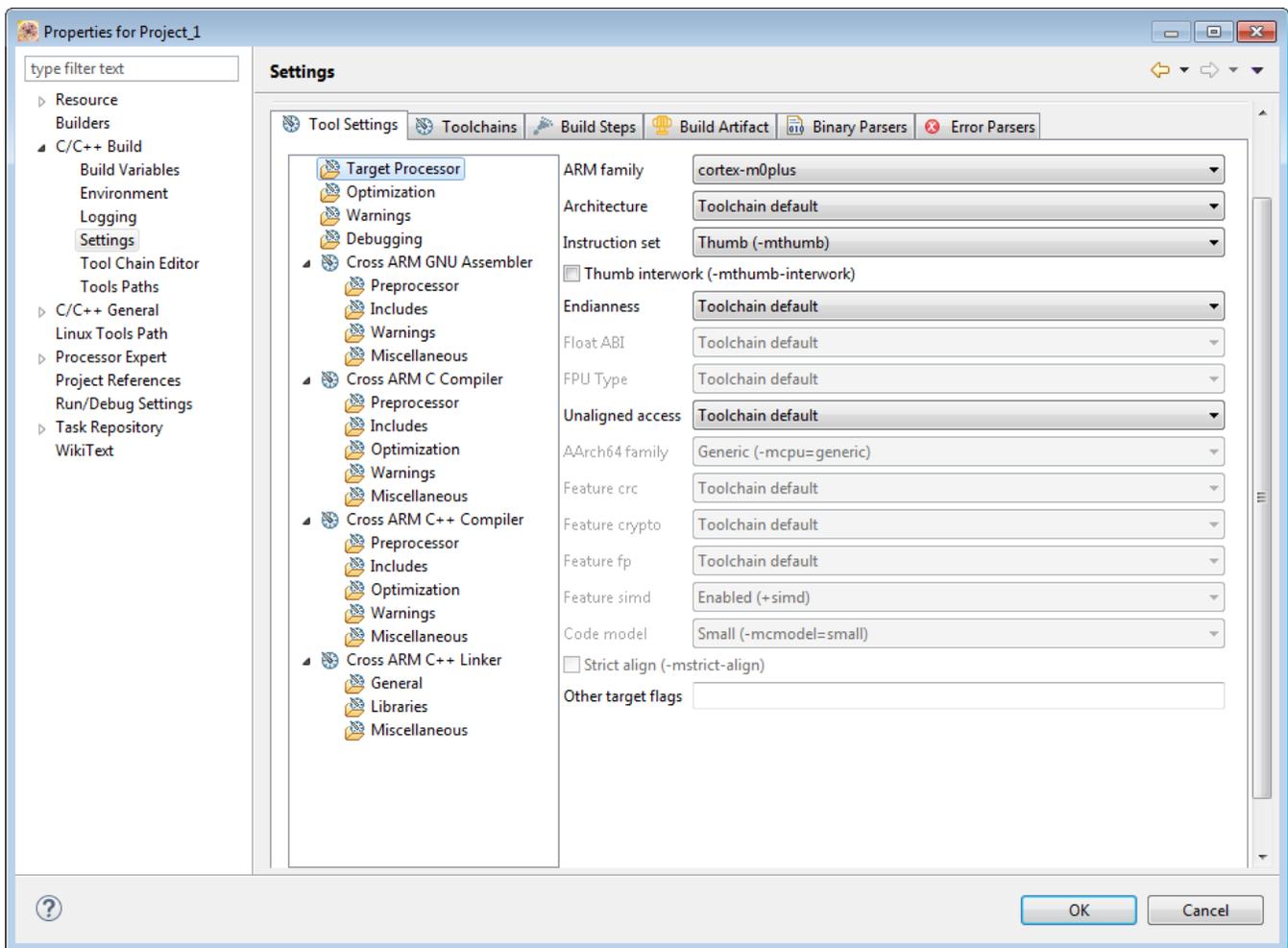
The Kinetis New Project wizard creates a set of build properties for the project. You can modify these build properties to better suit your needs.

Perform these steps to change build properties:

1. Start the IDE.
2. In the **Kinetis Design Studio Projects** view, select the project for which you want to modify the build properties.
3. Select **Project > Properties**.

The **Properties** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.

4. Expand the **C/C++ Build** property.
5. Select **Settings**.



The **Properties** window shows the corresponding build properties.

6. Use the Configuration drop-down list to specify the launch configuration for which you want to modify the build properties.
7. Click the **Tool Settings** tab.

The corresponding page appears.

8. From the list of tools on the **Tool Settings** page, select the tool for which you want to modify properties.
9. Change the settings that appear in the page.
10. Click **Apply** .

The IDE saves your new settings. You can select other tool pages and modify their settings.

11. When you finish, click OK to save your changes and close the **Properties** window.
12. In the Project Explorer view, right click on the project and select Clean Project. Once cleaned select Build Project. Monitor the generated command lines used to build the embedded application in the build Console view. Any problems with the build will

be reported under the Problems view. Assuming the build is successful, the generated binary will be listed under the project in the Project Explorer view.

3.2 Restoring Build Properties

If you modify a build configuration that the new project wizard generates, you can restore that configuration to its default state. You might want to restore the build properties in order to have a factory-default configuration, or to revert to a last-known working build configuration. To undo your modifications to build properties, click the **Restore Defaults** button at the bottom of the **Properties** window.

3.3 Defining C/C++ Build Settings and Behavior

The **C/C++ Build** page includes all builder-specific property pages.

- [Define Build Settings](#)
- [Define Build Behavior](#)

NOTE

Modifying settings such as the **Generate makefiles automatically** option, might enable or disable some parameters in some situations and change the availability of other property pages.

3.3.1 Define Build Settings

To define build settings, perform these steps.

1. Start the IDE.
2. In the **Kinetis Design Studio Projects** view, select the project for which you want to modify the build settings.
3. Select **Project > Properties**.

The **Properties for <project>** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.

4. Select **C/C++ Build**.

The C/C++ Build page appears.

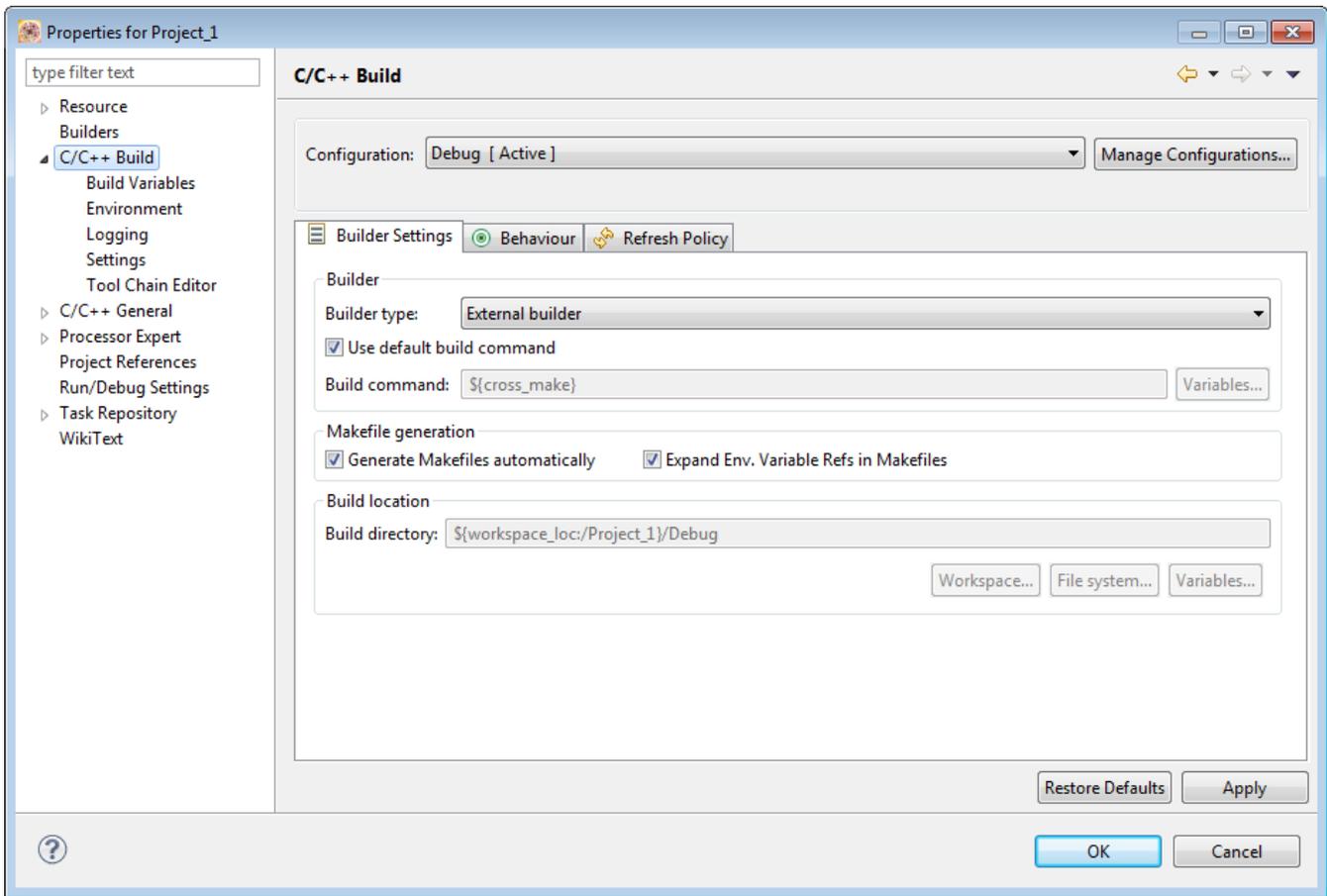


Figure 3-1. C/C++ Build Page - Builder Settings

5. Click the **Builder Settings** tab.

The builder settings for the selected build configuration appears. The table below describes the builder settings options.

Table 3-1. Builder Settings Options

Group	Option	Description
	Configuration	Specifies the type of configurations for the selected project.
	Manage configurations	Click to open the Manage Configurations dialog box that lets you set configurations based on the specified toolchains of the selected project.

Table continues on the next page...

**Table 3-1. Builder Settings Options
(continued)**

Group	Option	Description
		You can also create new configurations, rename an existing configuration, or remove the ones that are no longer required.
Builder	Builder type	Specifies the type of builder to use: <ul style="list-style-type: none"> • Internal builder - Builds C/C++ programs using a compiler that implements the C/C++ Language Specifications. • External builder - External tools let you configure and run programs and Ant buildfiles using the Workbench, which can be saved and run at a later time to perform a build.
Builder	Use default build command	Check to indicate that you want to use the default make command. Clear when you want to use a new make command. This option is only available when the Builder type option is set to External.
Builder	Build command	Specifies the default command used to start the build utility for your specific toolchain. Use this field if you want to use a build utility other than the default make command.
Builder	Variables	Click to open the Select build variable dialog box and add the desired environment variables and custom variables to the build command.
Makefile generation	Generate Makefiles automatically	Check to enable Eclipse change between two different CDT modes: it either uses the customer's makefile for the build, if one exists, or it generates makefiles for the user.
Makefile generation	Expand Env. Variable Refs in Makefiles	Check to define whether environment variables should be expanded in makefile.
Build location	Build directory	Specifies the location where the build operation takes place. This location will contain the

Table continues on the next page...

**Table 3-1. Builder Settings Options
(continued)**

Group	Option	Description
		generated artifacts from the build process. This option appears disabled when the Generate Makefiles automatically option is enabled.
Build location	Workspace	Click to open the Folder Selection dialog box and select a workspace location for the project. This is the directory that will contain the plug-ins and features to build, including any generated artifacts.
Build location	File system	Click to open the Browse For Folder dialog box and select a folder.
Build location	Variables	Click to open the Select build variable dialog box and select a variable to specify as an argument for the build directory, or create and configure simple build variables which you can reference in build configurations that support variables.

6. Make the desired changes and click **OK**.

The **Properties for <project>** window will close.

3.3.2 Define Build Behavior

To define build behavior, perform these steps.

1. Start the IDE.
2. In the **Kinetis Design Studio Projects** view, select the project for which you want to modify the build settings.
3. Select **Project > Properties**.

The **Properties** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.

4. Select **C/C++ Build**.

The **C/C++ Build** page appears.

5. Click the **Behaviour** tab.

The behavior settings for the selected build configuration appears.

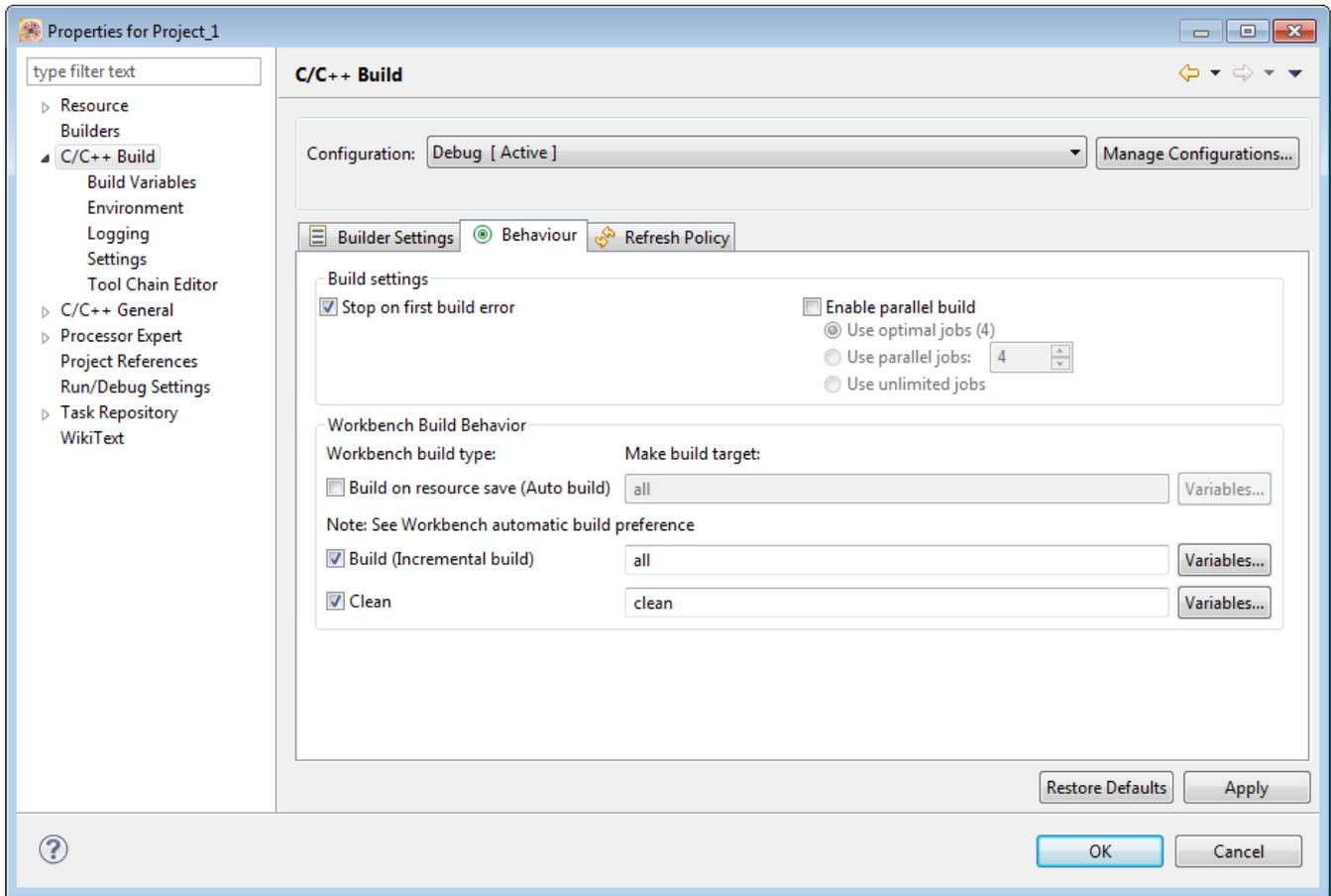


Figure 3-2. C/C++ Build Page - Behaviour

The table below describes the builder settings options.

Table 3-2. Behavior Options

Group	Option	Description
Build settings	Enable project specific settings	Check if you want to enable project specific settings.
Build settings	Stop on first build error	Check to stop building when Eclipse encounters an error. Clearing this option is helpful for building large projects as it enables make to continue making other independent rules even when one rule fails.
Build settings	Enable parallel build	Check to activate the generation of parallel builds to reduce the

Table continues on the next page...

**Table 3-2. Behavior Options
(continued)**

Group	Option	Description
		amount of time to build projects. The more cores your host processor has, the faster it will be. However, you need to determine the number of parallel jobs to perform: <ul style="list-style-type: none"> • Use optimal jobs number - Lets the system determine the optimal number of parallel jobs to perform. • Use parallel jobs - Lets you specify the maximum number of parallel jobs to perform. • Use unlimited jobs - Lets the system perform unlimited jobs.
Workbench Build Behavior	Workbench build type	Specifies the builder settings when instructed to build, rebuild, and clean.
Workbench Build Behavior	Build on resource save (Auto build)	Check to build your project whenever resources are saved. By default, this option is selected and builds occur automatically each time resources are modified. Clear if you do want that the build occurs only manually using a menu item.
Workbench Build Behavior	Build (Incremental Build)	Defines what the standard builder will call when an incremental build is performed.
Workbench Build Behavior	Variables	Click to open the Select build variable dialog box and add variables to the make build target command.
Workbench Build Behavior	Clean	Defines what the standard builder calls when a clean is performed. The make clean is defined in the makefile.
Workbench Build Behavior	Variables	Click to open the Select build variable dialog box and add variables to the make build target command.

6. Make the desired changes and click **OK**.

The **Properties for <project>** window will close.

3.4 Configuring Build Properties

The **Properties for <project>** window shows the corresponding build properties for the project.

The properties that you specify in the **Tool Settings** panels apply to the selected build tool on the **Tool Settings** page of the **Properties for <project>** window.

The following table lists and describes the settings.

Table 3-3. Build Properties

Build Tool	Build Properties Panels
Target Processor	Target Processor
Optimization	Optimization
Warnings	Warnings
Debugging	Debugging
Cross ARM GNU Assembler	Cross ARM GNU Assembler > Preprocessor
	Cross ARM GNU Assembler > Includes
	Cross ARM GNU Assembler > Warnings
	Cross ARM GNU Assembler > Miscellaneous
Cross ARM C Compiler	Cross ARM C Compiler > Preprocessor
	Cross ARM C Compiler > Includes
	Cross ARM C Compiler > Optimization
	Cross ARM C Compiler > Warnings
	Cross ARM C Compiler > Miscellaneous
Cross ARM C++ Compiler	Cross ARM C++ Compiler > Preprocessor
	Cross ARM C++ Compiler > Includes
	Cross ARM C++ Compiler > Optimization
	Cross ARM C++ Compiler > Warnings
	Cross ARM C++ Compiler > Miscellaneous
Cross ARM C++ Linker	Cross ARM C++ Linker > General
	Cross ARM C++ Linker > Libraries
	Cross ARM C++ Linker > Miscellaneous

3.4.1 Target Processor

The following table lists the options in the **Target Processor** panel.

Table 3-4. Target Processor options

Option	Description
ARM family	Use to specify the ARM family name. Use 'cortex-m0plus' for all Kinetis-L and 'cortex-m4' for all other Kinetis devices.
Architecture	Use to specify the target hardware architecture or processor name. The compiler can take advantage of the extra instructions that the selected architecture provides and optimize the code to run on a specific processor. The inline assembler might display error messages or warnings if it assembles some processor-specific instructions for the wrong target architecture. Default: Toolchain default
Instruction set	Use to generate suitable interworking veneers when it links the assembler output. You must enable this option if you write ARM code that you want to interwork with Thumb code or vice versa. The only functions that need to be compiled for interworking are the functions that are called from the other state. You must ensure that your code uses the correct interworking return instructions. Default: Thumb (-mthumb)
Thumb interwork (-mthumb-interwork)	Check to have the processor generate Thumb code instructions. Clear to prevent the processor from generating Thumb code instructions. The IDE enables this setting only for architectures and processors that support the Thumb instruction set. Default: Clear
Endianness	Use to specify the byte order of the target hardware architecture: <ul style="list-style-type: none"> • Little-little endian; right-most bytes (those with a higher address) are most significant • Big-big endian; left-most bytes (those with a lower address) are most significant Default: Toolchain default. Use 'FP instructions (hard)' for devices with hardware floating point unit (Cortex-M4F devices)
Float ABI	Use to specify the float Application Binary Interface (ABI). Default: Toolchain default
FPU Type	Use to specify the type of floating-point unit (FPU) for the target hardware architecture: The assembler might display error messages or warnings if the selected FPU architecture is not compatible with the target architecture. Default: Toolchain default. Use 'fpv4-sp-d16' for Kinetis devices with hardware floating point unit (Cortex-M4F devices).
Unaligned access	Use to specify unaligned access. Default: Toolchain default
AArch64 family	Use to specify the architecture family: <ul style="list-style-type: none"> • Generic (-mcpu=generic) • Large (-mcpu=large) • Toolchain default Default: Toolchain default
Feature crc	Use to specify Feature crc.
Feature crypts	Use to specify Feature crypts.
Feature fp	Use to specify Feature fp.
Feature simd	Use to specify Feature simd.
Code model	Specifies the addressing mode that the linker uses when resolving references. This setting is equivalent to specifying the -mcmode keyword command-line option. <ul style="list-style-type: none"> • Tiny (-mcmode=tiny)

Table continues on the next page...

Table 3-4. Target Processor options (continued)

Option	Description
	<ul style="list-style-type: none"> • Small (-mcmmodel=small) • Large (-mcmmodel=large) • Toolchain default
Strict align (-mstrict-align)	Controls the use of non-standard ISO/IEC 9899-1990 ("C90") language features.
Other target flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI.

3.4.2 Optimization

The following table lists the options in the **Optimization** panel.

Table 3-5. Optimization options

Option	Description
Optimization level	<p>Specify the optimizations that you want the compiler to apply to the generated object code:</p> <ul style="list-style-type: none"> • None (-O0) - Disable optimizations. This setting is equivalent to specifying the -O0 command-line option. The compiler generates unoptimized, linear assembly-language code. • Optimize (-O1) - The compiler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. This setting is equivalent to specifying the -O1 command-line option. The compiler omits all target-specific optimizations and generates linear assembly-language code. • Optimize more (-O2) - The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the -O2 command-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code. • Optimize most (-O3) - The compiler performs all the level 2 optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the that is usually faster than the code generated from level 2 optimizations. • Optimize size (-Os) - The compiler optimizes object code at the specified Optimization Level such that the resulting binary file has a smaller executable code size, as opposed to a faster execution speed. This setting is equivalent to specifying the -Os command-line option. • Optimize for debugging (-Og) - The compiler optimizes object code at the specified Optimization Level such that the resulting binary file has a faster execution speed, as opposed to a smaller executable code size.
Message length (-fmessage-length=0)	Check if you want to specifies the maximum length in bytes for the message.
'char' is signed (-fsigned-char)	Check to treat char declarations as signed char declarations.
Function sections (-ffunction-sections)	Check to enable function sections.
Data sections (-fdata-sections)	Check to enable data sections.
Do not use _cxa_atexit() (-fnouse-cxa-atexit)	Check if you do not want to use _cxa_atexit().

Table continues on the next page...

Table 3-5. Optimization options (continued)

Option	Description
Single precision constants (-fsingle-precision-constant)	Check to enable single precision constants.
Position independent code (-fPIC)	Select to instruct the build tools to generate position independent-code.
Other optimization flags	Specify additional command line options; type in custom optimization flags that are not otherwise available in the UI.

3.4.3 Warnings

The following table lists the options in the **Warnings** panel.

Table 3-6. Warnings options

Option	Description
Check syntax only (-fsyntax-only)	Check this option if if you want to check the syntax of commands and throw a syntax error.
Pedantic (-pedantic)	Check if you want warnings like -pedantic, except that errors are produced rather than warnings.
Pedantic warnings as errors (-pedantic-errors)	Check this option if if you want to inhibit the display of warning messages.
Inhibit all warnings (-w)	Check this option if if you want to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Enable all common warnings (-Wall)	Check this option if you want to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Extra warnings (-Wextra)	Check this option to enable any extra warnings.
Warn on implicit conversions (-Wconversion)	Check to warn of implicit conversions.
Warn on uninitialized variables (-Wuninitialised)	Check to warn of uninitialized variables.
Warn if floats are compared as equal (-Wfloat-equal)	Check to warn if floats are compared as equal.
Warn if shadowed variable (-Wshadow)	Check to warn if shadowed variable are used.
Warn if pointer arithmetic (-Wpointer-arith)	Check to warn if pointer arithmetic are used.
Warn if wrong cast (-Wbad-function-cast)	Check to warn of wrong cast.
Warn if suspicious logical ops (-Wlogical-op)	Check to warn in case of suspicious logical operation.
Warn in struct is returned (-Waggrate-return)	Check to warn if struct is returned.

Table continues on the next page...

Table 3-6. Warnings options (continued)

Option	Description
Warn on undeclared global function (-Wmissing-declaration)	Check to warn if an undeclared global function is encountered.
Generate errors instead of warnings (-Werror)	Check to generate errors instead of warnings.
Other warning flags	Specify additional command line options; type in custom warning flags that are not otherwise available in the UI.

3.4.4 Debugging

The following table lists the options in the **Debugging** panel.

Table 3-7. Debugging options

Option	Description
Debug level	Specify the debug levels: <ul style="list-style-type: none"> • None - No Debug level. • Minimal (-g1) - The compiler provides minimal debugging support. • Default (-g) - The compiler generates DWARF 1.x conforming debugging information. • Maximum (-g3) - The compiler provides maximum debugging support.
Debug format	Specify the debug formats for the compiler. Use Toolchain default.
Generate prof information (-p)	Generates extra code to write profile information suitable for the analysis program prof. You must use this option when compiling the source files you want data about, and you must also use it when linking.
Generate gprof information (-pg)	Generates extra code to write profile information suitable for the analysis program gprof. You must use this option when compiling the source files you want data about, and you must also use it when linking. Profiling requires special GNU libraries enabled for profiling. At the current time the GNU ARM Embedded (launchpad) libraries do not include this.
Other debugging flags	Specify additional command line options; type in custom debuggong flags that are not otherwise available in the UI.

3.4.5 Cross ARM GNU Assembler

The following table lists the options in the **Cross ARM GNU Assembler** panel.

Table 3-8. Cross ARM GNU Assembler options

Option	Description
Command	Shows the location of the assembler executable file. Default: <code>\${cross_prefix}\${cross_c}\${cross_suffix}</code>

Table continues on the next page...

Table 3-8. Cross ARM GNU Assembler options (continued)

Option	Description
All Options	Shows the actual command line the assembler will be called with. Default: <code>-x assembler-with-cpp -Xassembler -g</code>
Expert settings	
Command line patterns	Shows the expert settings command line parameters. Default: <code>\${COMMAND} \${cross_toolchain_flags} \${FLAGS} -c \${OUTPUT_FLAG} \${OUTPUT_PREFIX}\${OUTPUT} \${INPUTS}</code>

3.4.5.1 Cross ARM GNU Assembler > Preprocessor

The following table lists the options in the **Cross ARM GNU Assembler Preprocessor** panel.

Table 3-9. Cross ARM GNU Assembler Preprocessor options

Option	Description
Use preprocessor	Check this option to use the preprocessor for the assembler.
Do not search system directories (-nostdinc)	Check this option if you do not want the assembler to search the system directories. By default, this checkbox is clear. The assembler performs a full search that includes the system directories.
Preprocess only (-E)	Check this option if you want the assembler to preprocess source files and not to run the compiler. By default, this checkbox is clear and the source files are not preprocessed.
Defined symbols (-D)	Use this option to specify the substitution strings that the assembler applies to all the assembly-language modules in the build target. Enter just the string portion of a substitution string. The IDE prepends the -D token to each string that you enter. For example, entering <code>opt1 x</code> produces this result on the command line: <code>-Dopt1 x</code> . Note: This option is similar to the <code>DEFINE</code> directive, but applies to all assembly-language modules in a build target.
Undefined symbols (-U)	Undefined the substitution strings you specify in this panel.

3.4.5.2 Cross ARM GNU Assembler > Includes

The following table lists the options in the **Cross ARM GNU Assembler Includes** panel.

Table 3-10. Cross ARM GNU Assembler Includes options

Option	Description
Include paths (-I)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search <code>#include</code> files in several different ways. You can also set the search order as follows: For include statements of the form <code>#include"xyz"</code> , the compiler first searches user paths, then the system paths For include statements of the form <code>#include<xyz></code> , the compiler searches only system paths. This option is global.

Table continues on the next page...

Table 3-10. Cross ARM GNU Assembler Includes options (continued)

Option	Description
Include files (-include)	Use this option to specify the include file search path.

3.4.5.3 Cross ARM GNU Assembler > Warnings

The following table lists the options in the **Cross ARM GNU Assembler Warnings** panel.

Table 3-11. Cross ARM GNU Assembler Warnings options

Option	Description
Other warning flags	Specify additional command line options; type in custom warning flags that are not otherwise available in the UI.

3.4.5.4 Cross ARM GNU Assembler > Miscellaneous

The following table lists the options in the **Cross ARM GNU Assembler Miscellaneous** panel.

Table 3-12. Cross ARM GNU Assembler Miscellaneous options

Option	Description
Assembler flags	Specify the flags that need to be passed with the assembler.
Generates assembler listing (-Wa, -adhlns="\$@.lst")	Enables the assembler to create a listing file as it compiles assembly language into object code.
Verbose (-v)	Check this option if you want the IDE to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the -v command-line option. By default this checkbox is clear. The IDE displays just error messages that the compiler emits. The IDE suppresses warning and informational messages.
Other flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI.

3.4.6 Cross ARM C Compiler

The following table lists the options in the **Cross ARM C Compiler** panel.

Table 3-13. Cross ARM C Compiler options

Option	Description
Command	Shows the location of the compiler executable file. Default: <code>\${cross_prefix}\${cross_c}\${cross_suffix}</code>
All Options	Shows the actual command line the compiler will be called with.
Command line patterns	Shows the expert settings command line parameters. Default: <code>\${COMMAND} \${cross_toolchain_flags} \${FLAGS} -c \${OUTPUT_FLAG} \${OUTPUT_PREFIX}\${OUTPUT} \${INPUTS}</code>

3.4.6.1 Cross ARM C Compiler > Preprocessor

The following table lists the options in the **Cross ARM C Compiler Preprocessor** panel.

Table 3-14. Cross ARM C Compiler Preprocessor options

Option	Description
Do not search system directories (-nostdinc)	Check this option if you do not want the compiler to search the system directories. By default, this checkbox is clear. The compiler performs a full search that includes the system directories.
Preprocess only (-E)	Check this option if you want the compiler to preprocess source files and not to run the compiler. By default, this checkbox is clear and the source files are not preprocessed.
Defined symbols (-D)	Use this option to specify the substitution strings that the compiler applies modules in the build target. Enter just the string portion of a substitution string. The IDE prepends the -D token to each string that you enter. For example, entering <code>opt1 x</code> produces this result on the command line: <code>-Dopt1 x</code> . Note: This option is similar to the <code>DEFINE</code> directive, but applies to all assembly-language modules in a build target.
Undefined symbols (-U)	Undefined the substitution strings you specify in this panel.

3.4.6.2 Cross ARM C Compiler > Includes

The following table lists the options in the **Cross ARM C Compiler Includes** panel.

Table 3-15. Cross ARM C Compiler Includes options

Option	Description
Include paths (-I)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search <code>#include</code> files in several different ways. You can also set the search order as follows: For include statements of the form <code>#include"xyz"</code> , the compiler first searches user paths, then the system paths For include statements of the form <code>#include<xyz></code> , the compiler searches only system paths This option is global.
Include files (-include)	Use this option to specify the include file search path.

3.4.6.3 Cross ARM C Compiler > Optimization

The following table lists the options in the **Optimization** panel.

Table 3-16. Cross ARM C Compiler Optimization options

Option	Description
Language standard	Select the programming language or standard to which the compiler should conform. <ul style="list-style-type: none"> • ISO C90 (-ansi) - Select this option to compile code written in ANSI standard C. The compiler does not enforce strict standards. For example, your code can contain some minor extensions, such as C++ style comments (//), and \$ characters in identifiers. • ISO C99 (-std=c99) - Select this option to instruct the compiler to enforce stricter adherence to the ANSI/ISO standard. • Compiler Default (ISO C90 with GNU extensions) - Select this option to enforce adherence to ISO C90 with GNU extensions. • ISO C99 with GNU Extensions (-std=gnu99)
Other optimization flags	Specify additional command line options; type in custom optimization flags that are not otherwise available in the UI.

3.4.6.4 Cross ARM C Compiler > Warnings

The following table lists the options in the **Warnings** panel.

Table 3-17. Cross ARM C Compiler Warnings options

Option	Description
Other warning flags	Specify additional command line options; type in custom warning flags that are not otherwise available in the UI.

3.4.6.5 Cross ARM C Compiler > Miscellaneous

The following table lists the options in the **Miscellaneous** panel.

Table 3-18. Cross ARM C Compiler Miscellaneous options

Option	Description
Generates assembler listing (-Wa, -adhlns="\$@.lst")	Enables the assembler to create a listing file as it compiles assembly language into object code.
Save temporary files (--save-temps Use with caution!)	Enables you to save temporary files.

Table continues on the next page...

Table 3-18. Cross ARM C Compiler Miscellaneous options (continued)

Option	Description
Verbose (-v)	Check this option if you want the IDE to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the -v command-line option. By default this checkbox is clear. The IDE displays just error messages that the compiler emits. The IDE suppresses warning and informational messages.
Other compiler flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI.

3.4.7 Cross ARM C++ Compiler

The following table lists the options in the **Cross ARM C++ Compiler** panel.

Table 3-19. Cross ARM C++ Compiler options

Option	Description
Command	Shows the location of the compiler executable file. Default: <code>\${cross_prefix}\${cross_cpp}\${cross_suffix}</code>
All Options	Shows the actual command line the compiler will be called with. Default: <code>-std=gnu++11 -fabi-version=0</code>
Expert settings	
Command line patterns	Shows the expert settings command line parameters. Default: <code>\${COMMAND} \${cross_toolchain_flags} \${FLAGS} -c \${OUTPUT_FLAG} \${OUTPUT_PREFIX}\${OUTPUT} \${INPUTS}</code>

3.4.7.1 Cross ARM C++ Compiler > Preprocessor

The following table lists the options in the **Cross ARM C Compiler Preprocessor** panel.

Table 3-20. Cross ARM C++ Compiler Preprocessor options

Option	Description
Do not search system directories (-nostdinc)	Check this option if you do not want the compiler to search the system directories. By default, this checkbox is clear. The compiler performs a full search that includes the system directories.
Do not search system C++ directories (-nostdinc++)	Check this option if you do not want the compiler to search the C++ system directories. By default, this checkbox is clear. The compiler performs a full search that includes the system directories.
Preprocess only (-E)	Check this option if you want the compiler to preprocess source files and not to run the compiler. By default, this checkbox is clear and the source files are not preprocessed.

Table continues on the next page...

Table 3-20. Cross ARM C++ Compiler Preprocessor options (continued)

Option	Description
Defined symbols (-D)	Use this option to specify the substitution strings that the compiler applies modules in the build target. Enter just the string portion of a substitution string. The IDE prepends the -D token to each string that you enter. For example, entering opt1 x produces this result on the command line: -Dopt1 x. Note: This option is similar to the DEFINE directive, but applies to all assembly-language modules in a build target.
Undefined symbols (-U)	Undefines the substitution strings you specify in this panel.

3.4.7.2 Cross ARM C++ Compiler > Includes

The following table lists the options in the Cross ARM C++ Compiler Includes panel.

Table 3-21. Cross ARM C++ Compiler Includes options

Option	Description
Include paths (-I)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search #include files in several different ways. You can also set the search order as follows: For include statements of the form #include"xyz", the compiler first searches user paths, then the system paths For include statements of the form #include<xyz>, the compiler searches only system paths This option is global.
Include files (-include)	Use this option to specify the include file search path.

3.4.7.3 Cross ARM C++ Compiler > Optimization

The following table lists the options in the **Optimization** panel.

Table 3-22. Cross ARM C++ Compiler Optimization options

Option	Description
Language standard	Select the programming language or standard to which the compiler should conform. <ul style="list-style-type: none"> • ISO C90 (-ansi) - Select this option to compile code written in ANSI standard C. The compiler does not enforce strict standards. For example, your code can contain some minor extensions, such as C++ style comments (//), and \$ characters in identifiers. • ISO C99 (-std=c99) - Select this option to instruct the compiler to enforce stricter adherence to the ANSI/ISO standard. • Compiler Default (ISO C90 with GNU extensions) - Select this option to enforce adherence to ISO C90 with GNU extensions. • ISO C99 with GNU Extensions (-std=gnu99)
ABI version	Enables you to select the Application Binary Interface (ABI) the compiler and assembler uses for function calls and structure layout.

Table continues on the next page...

Table 3-22. Cross ARM C++ Compiler Optimization options (continued)

Option	Description
Do not use exceptions (-fno-exceptions)	Check if you do not want to display exceptions.
Do not use RTTI (-fno-rtti)	Check if you do not want to use of the C++ runtime type information (RTTI) capabilities, including the dynamic_cast and typeid operators. Clear to let the compiler generate smaller, faster object code but do not allow runtime type information operations. The checkbox corresponds to the pragma RTTI and the command-line option -RTTI.
Other optimization flags	Specify additional command line options; type in custom optimization flags that are not otherwise available in the UI.

3.4.7.4 Cross ARM C++ Compiler > Warnings

The following table lists the options in the **Warnings** panel.

Table 3-23. Cross ARM C++ Compiler Warnings options

Option	Description
Warn on ABI violations (-Wabi)	Check if you want to display warnings in case of ABI violation.
Warn on class privacy (-Wctor-dtor-privacy)	Check if you want to display warnings about class privacy.
Warn on uncast NULL (-Wsing-null-sentinel)	Check if you want to display warnings on uncast NULL.
Warn on sign promotion (-Wsign-promo)	Check if you want to display warnings in case of sign promotion.
Warn about Effective C++ violations (-Weffc++)	Check if you want to display warnings in case of effective C++ violations.
Other warning flags	Specify additional command line options; type in custom warning flags that are not otherwise available in the UI.

3.4.7.5 Cross ARM C++ Compiler > Miscellaneous

The following table lists the options in the **Miscellaneous** panel.

Table 3-24. Cross ARM C++ Compiler Miscellaneous options

Option	Description
Generates assembler listing (-Wa, -adhlns="\$@.lst")	Enables the assembler to create a listing file as it compiles assembly language into object code.

Table continues on the next page...

Table 3-24. Cross ARM C++ Compiler Miscellaneous options (continued)

Option	Description
Save temporary files (--save-temps Use with caution!)	Enables you to save temporary files.
Verbose (-v)	Check this option if you want the IDE to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the -v command-line option. By default this checkbox is clear. The IDE displays just error messages that the compiler emits. The IDE suppresses warning and informational messages.
Other compiler flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI.

3.4.8 Cross ARM C++ Linker

The following table lists the options in the **Cross ARM C Linker** panel.

Table 3-25. Cross ARM C Linker options

Option	Description
Command	Shows the location of the linker executable file. Default: <code>\${cross_prefix}\${cross_c}\${cross_suffix}</code>
All Options	Shows the actual command line the assembler will be called with. Default: <code>-T "C:\Users\b14174\workspace1-nn\Project_1\Project_Settings\Linker_Files/ProcessorExpert.ld" -Xlinker --gc-sections -L"C:\Users\b14174\workspace1-nn"</code>
Expert settings	
Command line patterns	Shows the expert settings command line parameters. Default: <code>\${COMMAND} \${cross_toolchain_flags} \${FLAGS} \${OUTPUT_FLAG} \${OUTPUT_PREFIX}\${OUTPUT} \${INPUTS}</code>

3.4.8.1 Cross ARM C++ Linker > General

The following table lists the options in the General panel.

Table 3-26. General options

Option	Description
Script files (-T)	This option passes the -T argument to the linker file.
Do not use standard start files (-nostartfiles)	This option passes the -nostartfiles argument to the linker file. It does not allow the use of the standard start files.
Do not use default libraries (-nodefaultlibs)	This option passes the -nodefaultlibs argument to the linker file. It does not allow the use of the default libraries.
No startup or default libs (-nostdlib)	This option passes the -nostdlib argument to the linker file. It does not allow the use of startup or default libs.

Table continues on the next page...

Table 3-26. General options (continued)

Option	Description
Remove unused sections (-Xlinker --gc-sections)	This option passes the -Xlinker --gc-sections argument to the linker file. It removes the unused sections.
Print removed sections (-Xlinker --print-gc-sections)	This option passes the -Xlinker --print-gc-sections argument to the linker file. It prints the removed sections.
Omit all symbol information (-s)	This option passes the -s argument to the linker file. This option omits all symbol information.

3.4.8.2 Cross ARM C++ Linker > Libraries

The following table lists the options in the Libraries panel.

Table 3-27. Libraries options

Option	Description
Libraries (-l)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search #include files in several different ways. You can also set the search order as follows: For include statements of the form #include"xyz", the compiler first searches user paths, then the system paths For include statements of the form #include<xyz>, the compiler searches only system paths This option is global.
Library search path (-L)	Use this option to specify the include library search path.

NOTE

- The converter does not distinguish between Executable projects and Static Library projects. In the latter case the options for converting Newlib-nano and adding semihosting should be suppressed.
- For more details on Using newlib-nano: -specs=nano.specs U -specs=nosys.specs using newlib: -specs=nosys.specs, see <http://mcuoneclipse.com/2014/07/11/switching-arm-gnu-tool-chain-and-libraries-in-kinetis-design-studio/>.

3.4.8.3 Cross ARM C++ Linker > Miscellaneous

The following table lists the options in the Miscellaneous panel.

Table 3-28. Miscellaneous options

Option	Description
Other objects	This option lists paths that the linker searches for objects. The linker searches the paths in the order shown in this list.

Table continues on the next page...

Table 3-28. Miscellaneous options (continued)

Option	Description
Generate Map	This option specifies the map filename. Default: \$ {BuildArtifactFileName}.map
Cross Reference (-Xlinker --cref)	Check this option to instruct the linker to list cross-reference information on symbols. This includes where the symbols were defined and where they were used, both inside and outside macros.
Print link map (-Xlinker --printf-map)	Check this option to instruct the linker to print the map file.
Verbose (-v)	Check this option to show verbose information, including hex dump of program segments in applications; default setting
Other linker flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI.

Chapter 4 Appendices

4.1 Installing Kinetis SDK

To install Kinetis SDK:

1. Download the latest Kinetis SDK.
2. Double-click to launch the executable. The **Introduction** page of the Freescale Kinetis SDK install wizard appears.

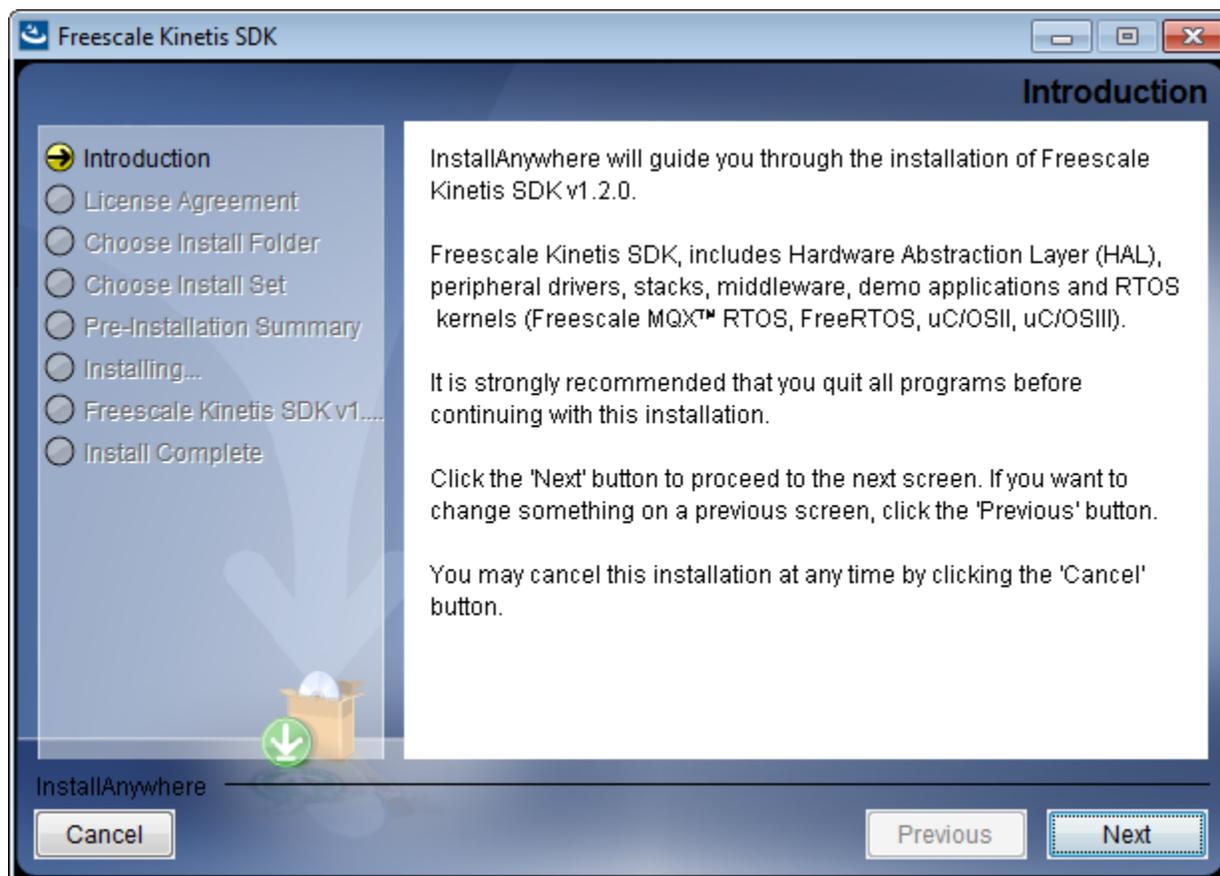


Figure 4-1. Launching the install wizard

3. Click **Next**. The **License Agreement** page of the wizard appears.

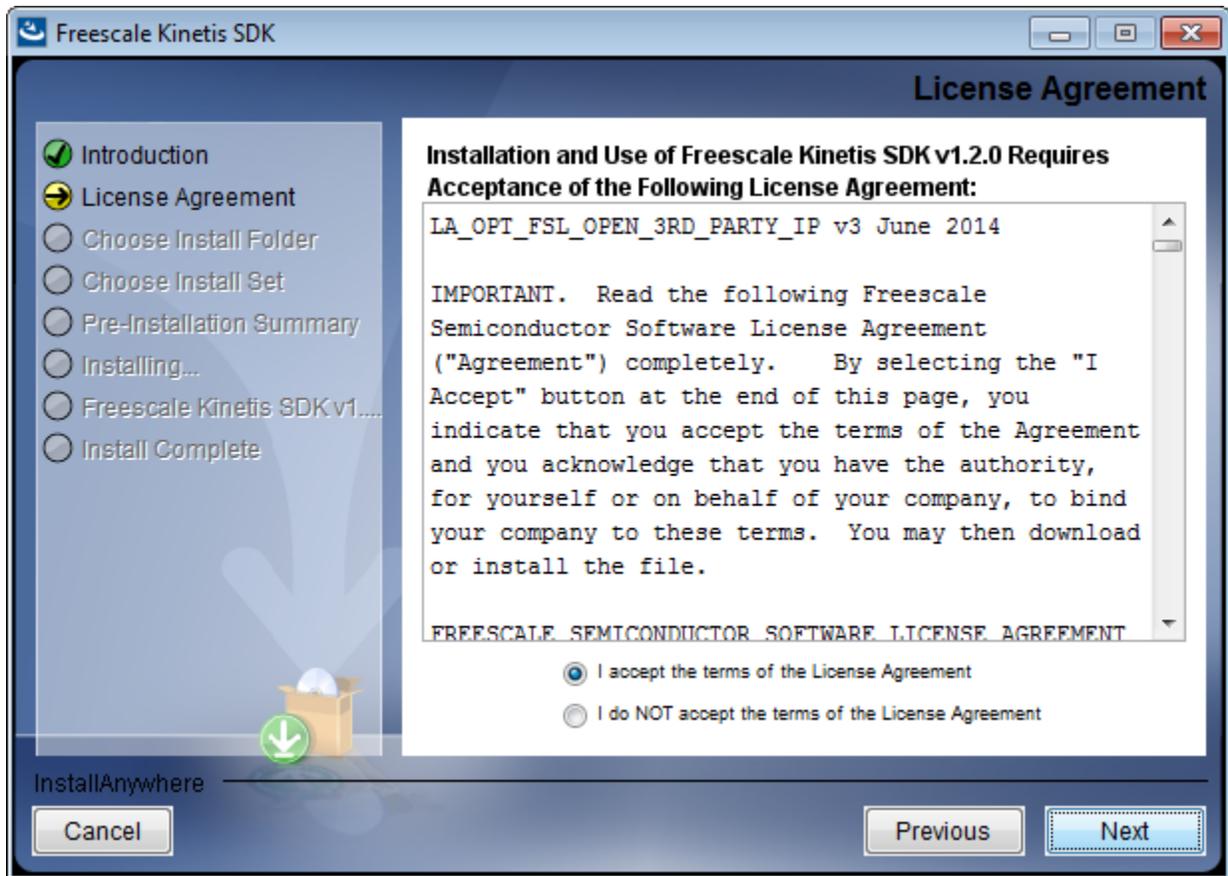


Figure 4-2. License installation

4. Accept the license and click **Next**. The Choose Install Folder page appears.
5. Select the location where you want to install KSDK.

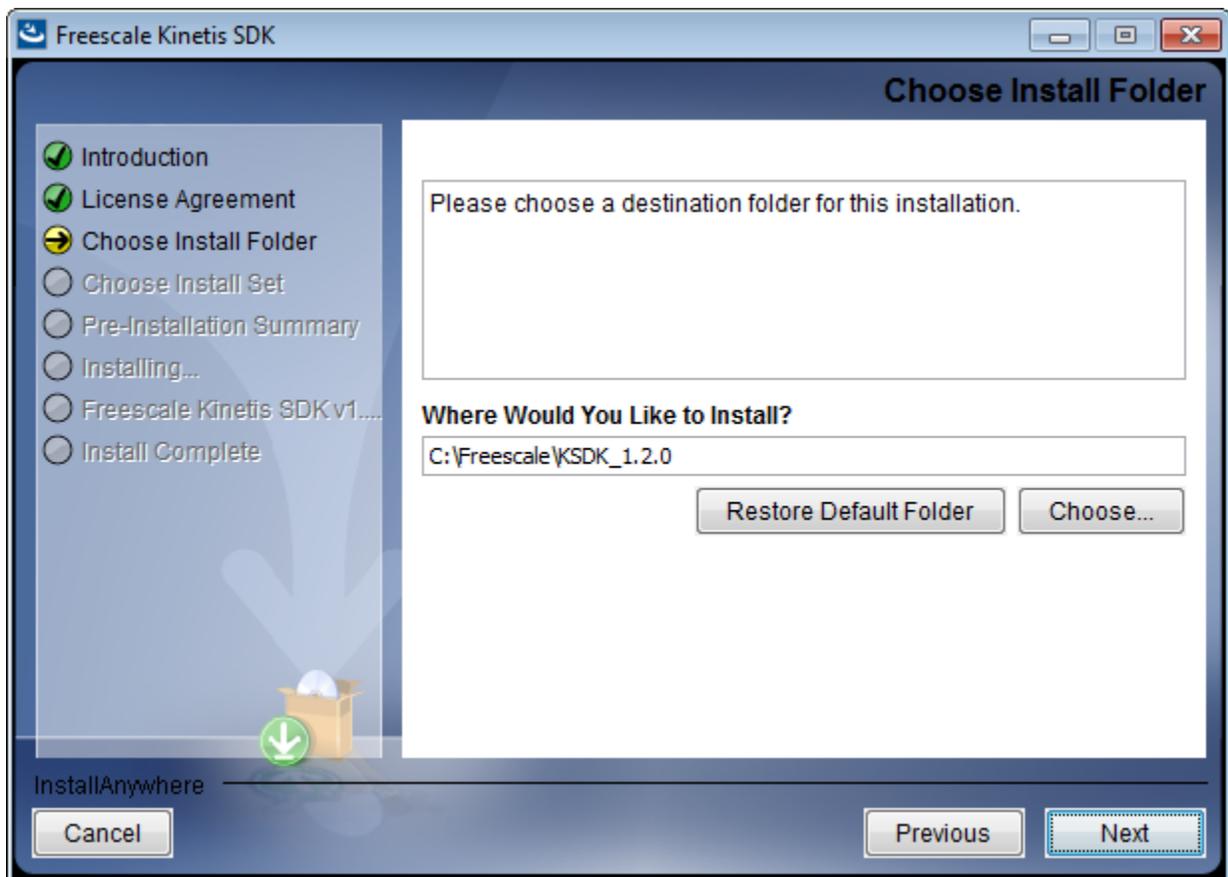


Figure 4-3. Select install folder

6. Click **Next**. The **Choose Install Set** page appears.
7. Select the install set based on your requirement. This scenario uses Kinetis SDK Basic.

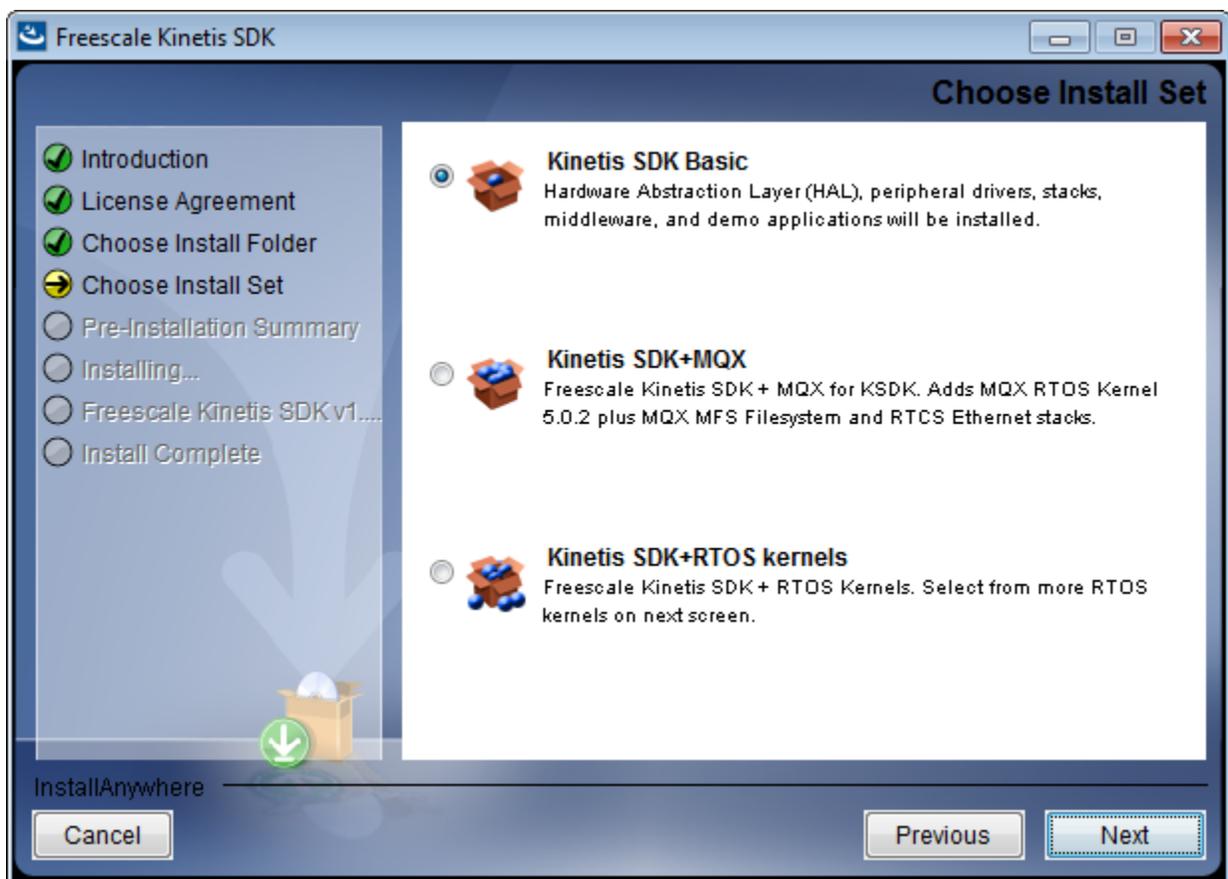


Figure 4-4. Choose install set

8. Click **Next**. The installation starts. The wizard prompts you on successful installation. The wizard also prompts if you want to check the Freescale Kinetis SDK release Notes.

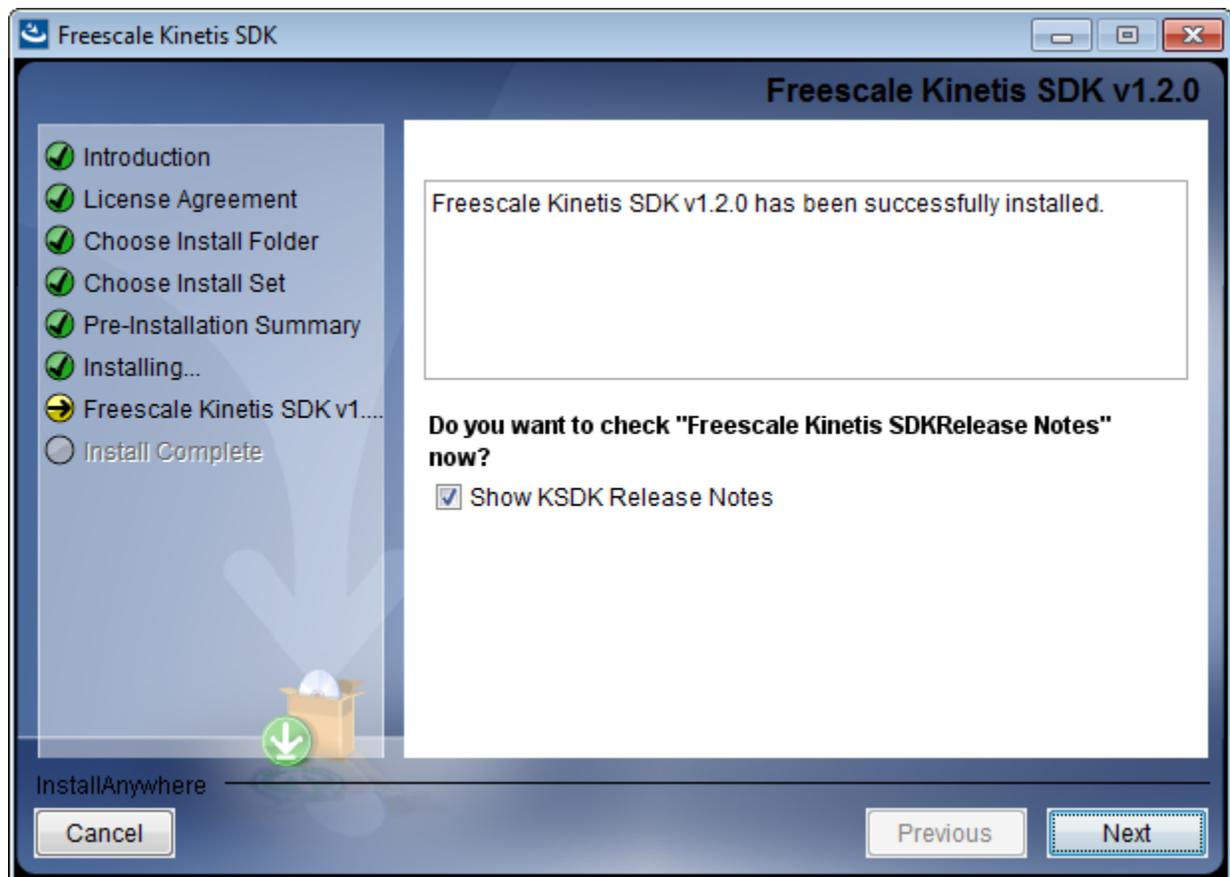


Figure 4-5. Installation successful

9. Click **Next**. The Install Complete page of the wizard appears.
10. Choose whether you immediately want to restart the system for completing the installation.

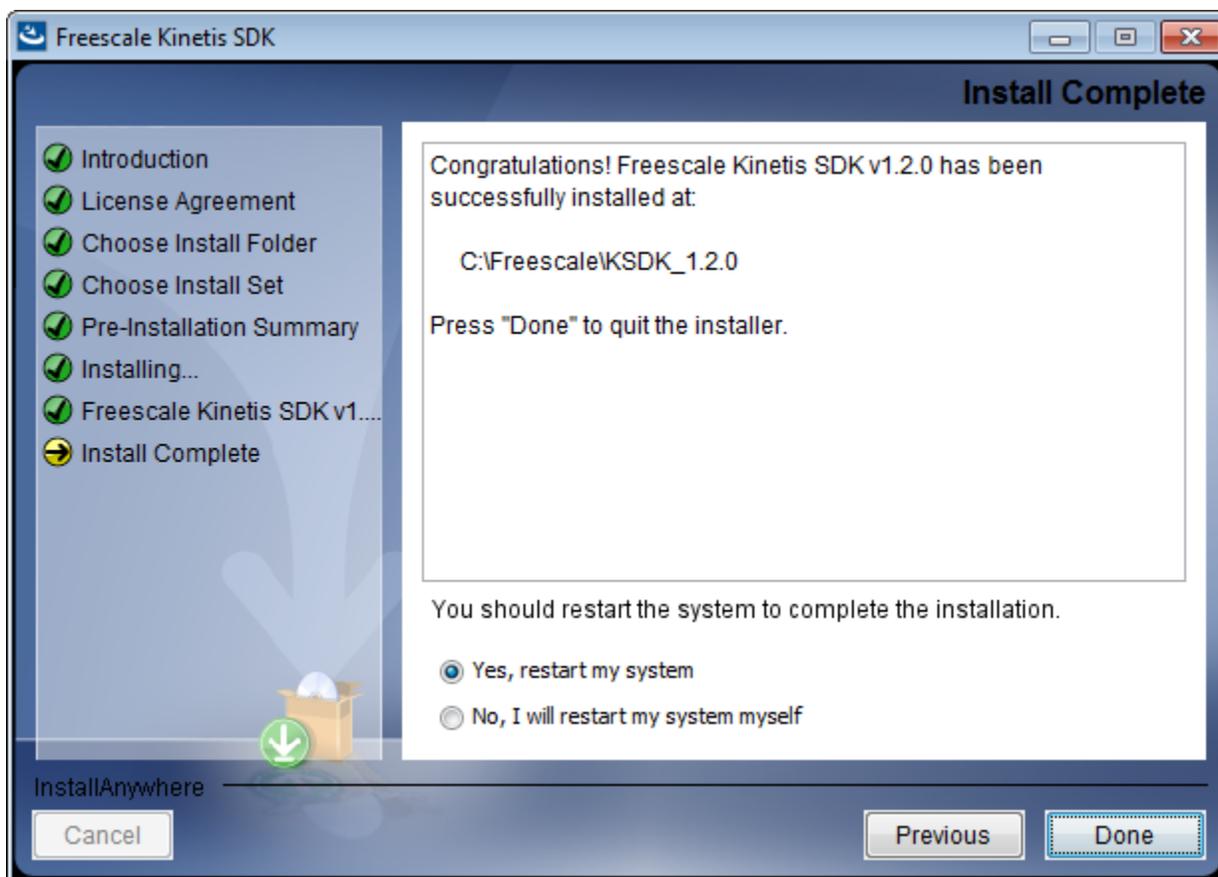


Figure 4-6. Installation complete

11. Click **Done** to close the installer.

4.2 Installing Kinetis SDK into KDS

To install Kinetis SDK in KDS:

1. Download and install the appropriate Kinetis SDK for your microcontroller. See <http://www.freescale.com/ksdk> for details.
2. The Kinetis SDK installation directory contains an Eclipse update. Use the Install New Software wizard to install the Eclipse update into in the Kinetis Design Studio.

NOTE

Users with the Kinetis Design Studio IDE installed in a read-only location, which is the default for Linux systems, must launch the Kinetis Design Studio IDE with administrative/root privileges to install the KSDK.

- a. Launch the Kinetis Design Studio.
- b. Select **Help > Install New Software** from the IDE menu bar.

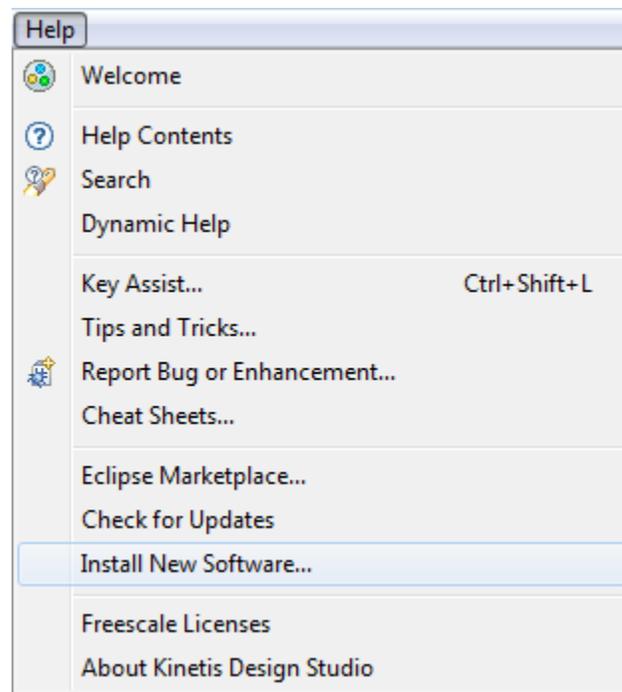


Figure 4-7. Install new software

The Available Software page of the Install wizard appears.

NOTE

You can also drag and drop the update file to the **Work with** text box in the dialog.

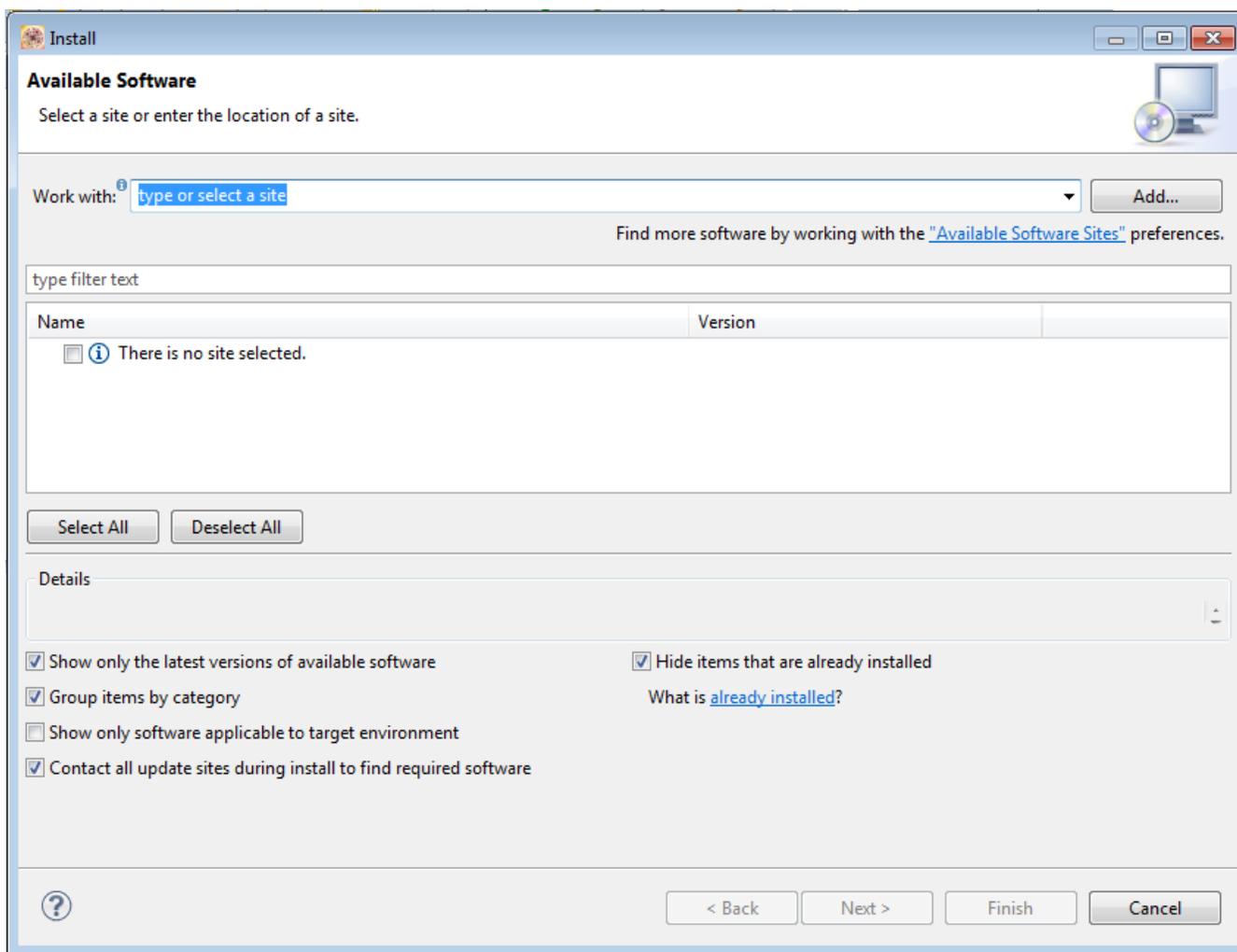


Figure 4-8. Available Software

c. Click **Add**. The **Add Repository** dialog appears.

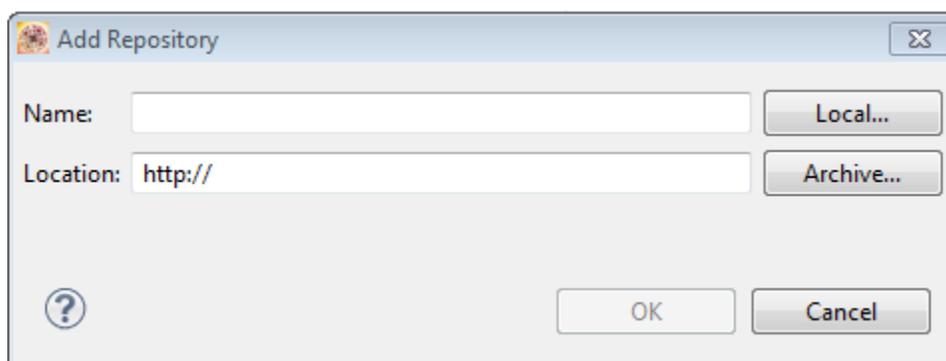


Figure 4-9. Add Repository dialog

- d. Click *Archive*.
- e. Select the .zip file provided by KSDK. For example, select `<DownloadDir> KSDK_1.2.0\tools\eclipse_update\KSDK_1.2.0_Eclipse_Update.zip`.
- f. Click **Open**.

- g. The Location text box is populated with the path you just suggested. You can add a name to identify the repository.

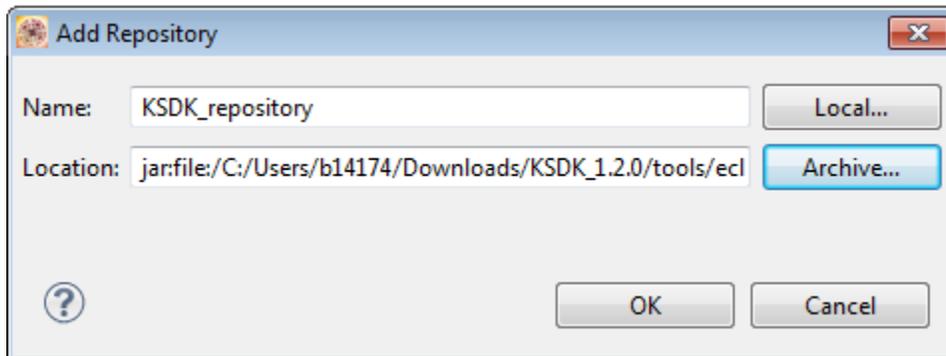


Figure 4-10. Location updated

- h. Click **OK**.
- i. The Work with text box is populated, and the list of available software is available.
- j. Check the checkbox to install Eclipse update for KSDK.

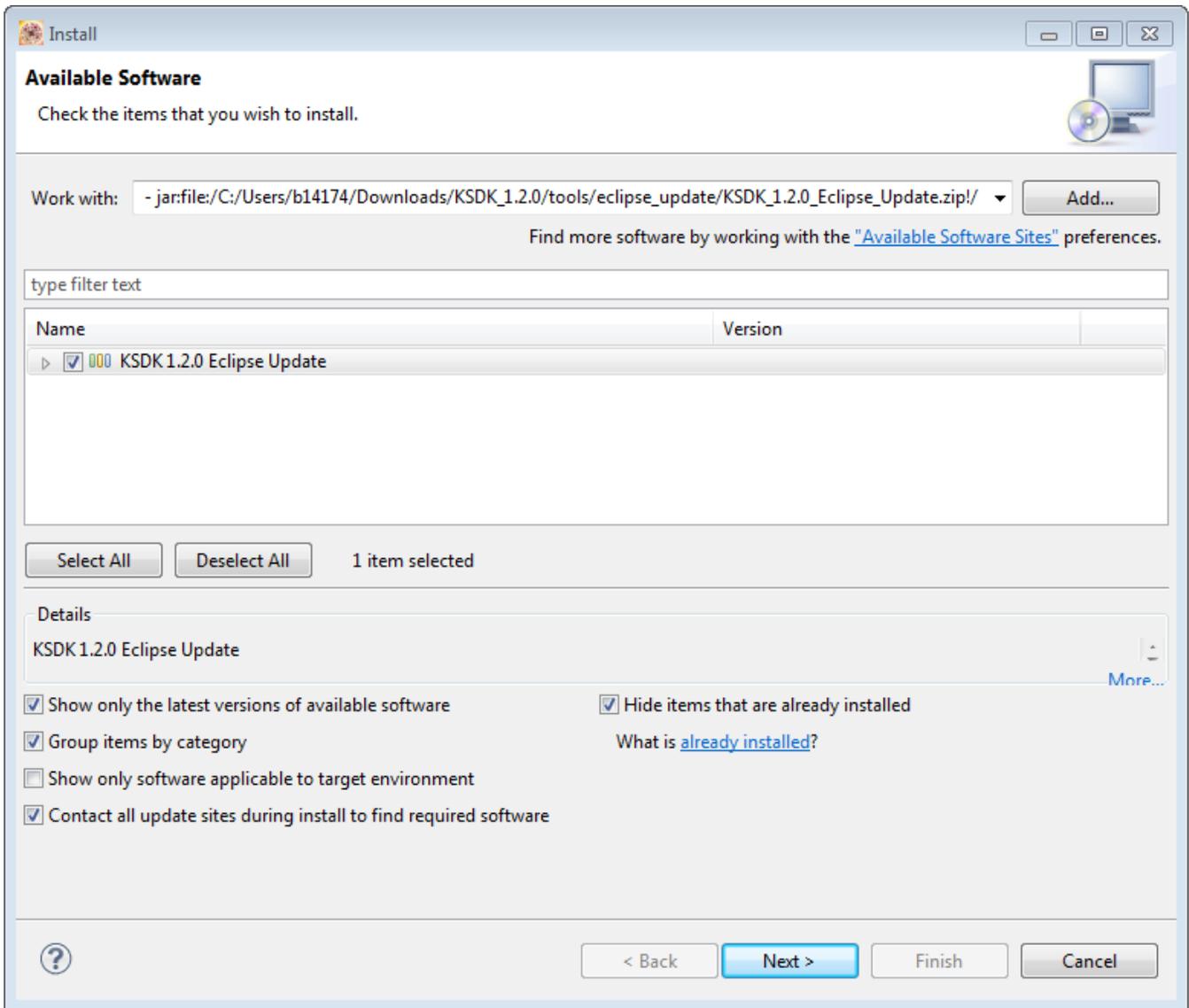


Figure 4-11. Select available software to install

- k. Click **Next**. The Install Details page of the wizard appears.
- l. Review the items in you want install.

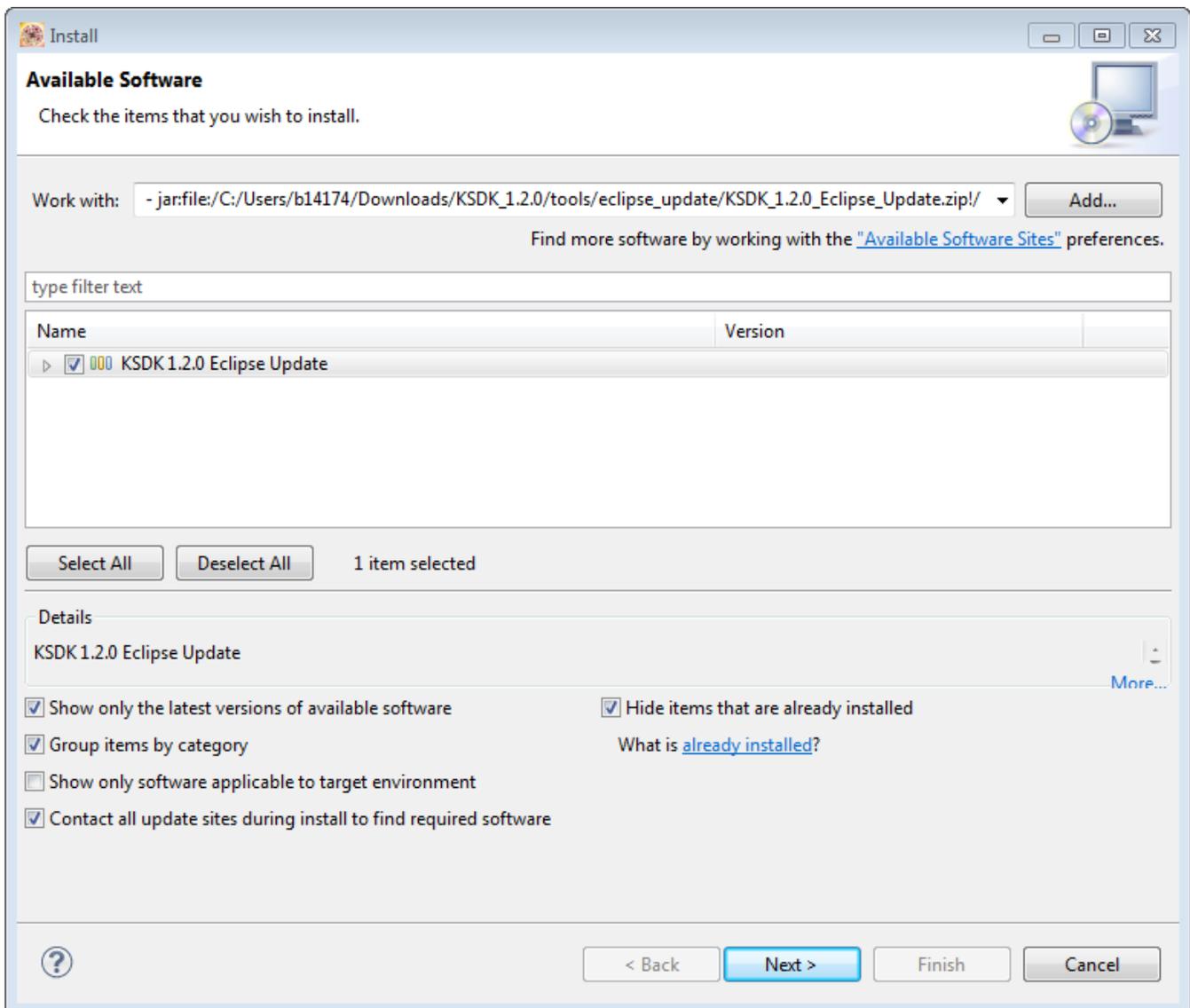


Figure 4-12. Review install items

- m. Click **Next**.
- n. Review and accept the terms of the license agreement.

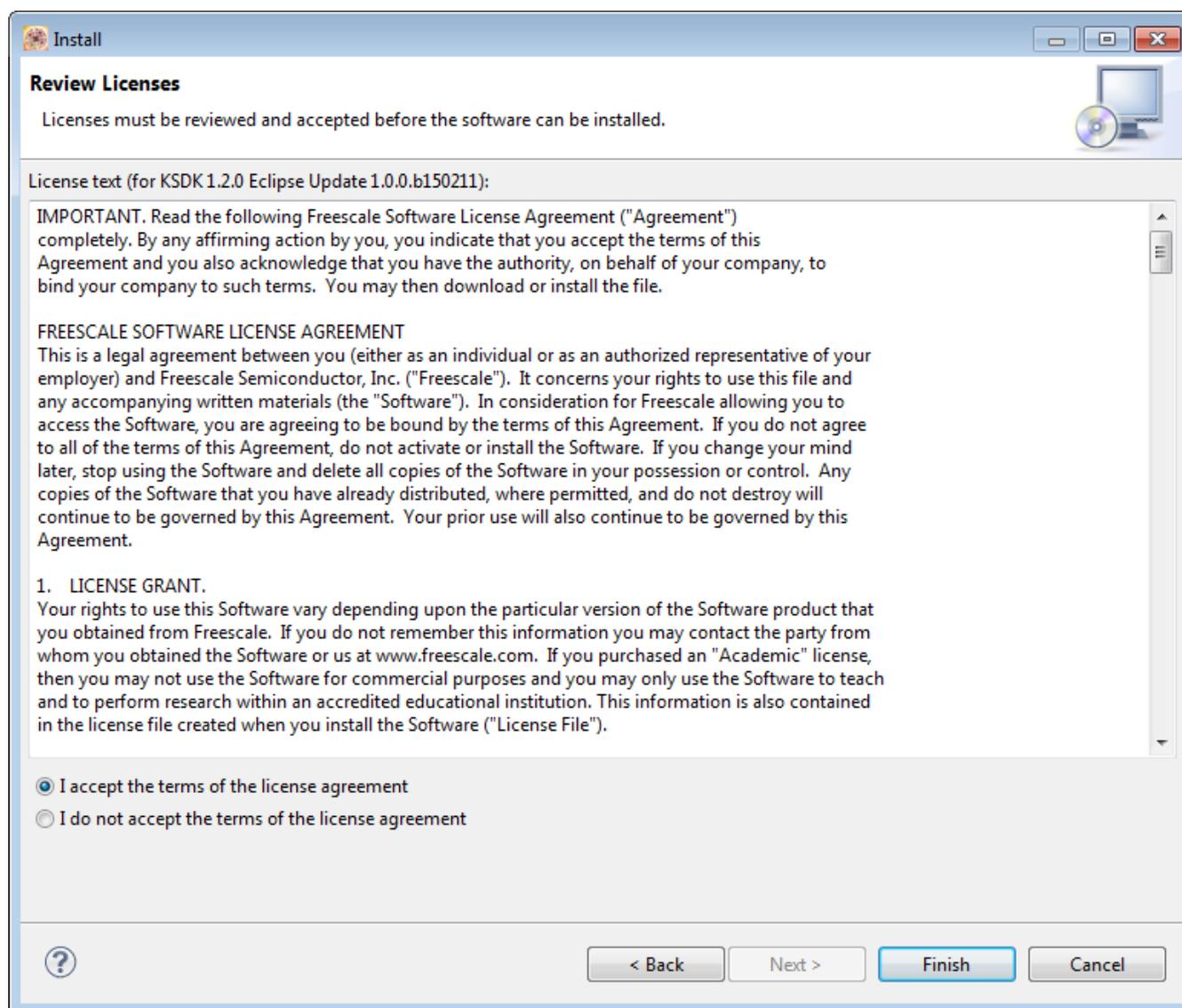


Figure 4-13. Review license

- o. Click **Finish**. The Kinetis SDK is installed into KDS.

4.3 Installing Drivers

When using either the OpenOCD, SEGGER J-Link or P&E Multilink debug interface, ensure that the relevant device drivers are installed.

For Windows installation

The Kinetis Design Studio Windows installer, by default, installs the SEGGER J-LINK and P&E Multilink device drivers.

However, if you want to manually install the device drivers:

- SEGGER J-Link:

Run the following installers:

- `install-dir\segger\USBDriver\InstDrivers.exe`
- `install-dir\segger\USBDriver\CDC\InstDriversCDC.exe`

- P&E Multilink driver:

Run the P&E driver installer:

`install-dir\pemicro\PEDrivers_install.exe`

- ARM mbed project Windows serial port driver:

Follow the instructions at:

For Linux installation

When the Kinetis Design Studio software development tools are installed on a Linux system, a udev rules file is included for each of the OpenOCD, SEGGER J-Link the P&E Multilink debug interfaces.

Table 4-1. Installing drivers

Debug interface	udev file location	Steps
Open OCD	<code>install-dir/openocd/ openocd.udev</code>	<ol style="list-style-type: none"> 1. Copy the udev file into the configuration directory (for example under <code>/etc/udev/ rules.d/</code>) 2. Rename the file to <code>99-openocd.rules</code> (for example) 3. Optionally the permissions allocated by the rules file can be adjusted. By default this requires users to be in the <code>plugdev</code> group. 4. Run the command <code>udevadm control --reload-rules</code> to instruct udev to reload its rules
Segger J-Link	<code>install-dir/segger/99- jlink.rules</code>	<ol style="list-style-type: none"> 1. Copy the udev file into the configuration directory (for example under <code>/etc/udev/ rules.d/</code>) 2. Run the command <code>udevadm control</code>
P&E Multilink	<code>install-dir/pemicro/drivers/ libusb_64_32/28-pemicro.rules</code>	Run the <code>setup.sh</code> script found under the same directory





How to Reach Us:

Home Page:

freescale.com

Web Support:

freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, Kinetis, and Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. ARM is the registered trademark of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All other product or service names are the property of their respective owners. All rights reserved.

© 2015, Freescale Semiconductor, Inc.

Document Number KDSUG
Revision 1.0, 04/2015

