
S32 Design Studio for S32 Platform 3.4

User Guide

Document Number: S32DSUG
Rev. 2.0, 04/2021



Contents

Part I: Quick Start Guide.....	6
Starting S32DS 3.4.....	7
Creating and building a project.....	8
Debugging a project.....	9
Quick links.....	11
Part II: Tasks.....	12
Installation management.....	13
Overview.....	13
Getting product updates.....	14
Getting updates automatically.....	14
Managing software sites.....	15
Downloading updates manually.....	16
Installing product updates and packages.....	16
Installing plug-ins.....	18
Viewing all installed software.....	20
Viewing installed updates and packages.....	21
Uninstalling packages.....	22
Uninstalling plug-ins.....	23
Uninstalling S32DS 3.4.....	24
License management.....	25
Overview.....	25
Licensing S32DS 3.4.....	25
Applying the license during installation.....	26
Getting the activation code.....	27
Viewing the license information.....	28
Viewing licenses on the website.....	28
Returning the license.....	29
Relicensing S32DS 3.4.....	30
Project management.....	31
Overview.....	31
Starting a project.....	32
Creating a project in the wizard.....	32
Creating a project from an example.....	34
Importing a project.....	35
Importing a project from a system folder or an archive file.....	35
Importing a project from a ProjectInfo.xml file.....	36
Adding files and folders to a project.....	36
Adding a device configuration.....	39
Creating a project with a device configuration.....	40
Editing a device configuration.....	40
Importing a device configuration.....	42
Exporting a device configuration.....	46
Locating project files and folders in the file system.....	47
Renaming a project.....	48
Duplicating a project.....	48
Saving a project to User Examples.....	49
Exporting a project.....	50

Exporting a project to a system folder or an archive file.....	50
Exporting a project to a ProjectInfo.xml file.....	51
Closing and reopening a project.....	52
Removing a project.....	53
Building projects.....	54
Overview.....	54
Using build configurations.....	55
Creating a build configuration.....	56
Setting the active build configuration.....	57
Editing a build configuration.....	57
Managing project resources in build configurations.....	61
Building a project.....	62
Resolving build problems.....	63
Adjusting the C/C++ indexer settings for large files.....	63
Building projects in non-English versions of Windows.....	66
Using optional tools.....	67
Generating an image file.....	67
Using output of optional tools in post-build steps.....	68
Preprocessing source files.....	72
Disassembling binaries and source files.....	73
Using parallel build.....	73
Debugging.....	74
Overview.....	74
Using the debugger.....	76
Using launch configurations.....	77
Creating a launch configuration.....	78
Editing a launch configuration.....	79
Running a launch configuration.....	87
Using launch groups.....	88
Creating a launch group.....	88
Running a launch group.....	90
Debugging on a bare-metal target.....	90
Selecting a hardware debug probe.....	91
Debugging with S32 Debug Probe from RAM.....	91
Debugging with S32 Debug Probe from flash for S32V23x targets.....	95
Debugging with S32 Debug Probe from flash for all other targets.....	100
Debugging with a PEMicro probe.....	103
Debugging with a Lauterbach probe.....	105
Viewing Registers.....	106
Viewing memory.....	115
Managing flash memory.....	116
Debugging on multiple cores.....	120
Debugging on a Linux target.....	123
Debugging on a VDK.....	126
Debugging Linux project on a VDK.....	127
Importing an executable.....	130
SDK management.....	131
Overview.....	131
Adding an SDK.....	131
Creating an SDK.....	132
Loading an SDK.....	134
Importing an SDK.....	134
Importing an MCAL SDK.....	136
Making a local SDK global.....	136
Using SDKs in projects.....	137
Attaching an SDK when creating a project.....	137

Attaching an SDK to an existing project.....	138
Upgrading SDK version.....	139
Detaching an SDK.....	139
Editing an SDK.....	140
Defining symbols.....	142
Exporting an SDK.....	143
Removing an SDK.....	143
Migration guide.....	144
Troubleshooting.....	146

Part III: Reference..... 150

User interface.....	151
Views and editors.....	151
Project Explorer view.....	151
Problems view.....	153
Breakpoints view.....	153
Debug view.....	154
Disassembly view.....	154
Expressions view.....	155
Memory view.....	155
Memory Browser view.....	156
Memory Spaces view.....	156
Registers view.....	158
EmbSys Registers view.....	158
Peripheral Registers view.....	160
Arm System Registers view.....	161
Watch registers view.....	162
Variables view.....	163
Intrinsics view.....	165
SDK Explorer view.....	165
Editor area.....	166
Wizards.....	167
New SDK wizard.....	167
Project creation wizards.....	169
Migrate wizard.....	178
Preferences.....	179
Perspectives.....	179
Available software sites.....	180
SDK Management.....	181
Project properties.....	183
SDKs.....	183
Perspectives.....	184
C/C++ perspective.....	184
Debug perspective.....	185
VDK Debug perspective.....	186
Git perspective.....	188
Build configuration.....	190
Build Tool Settings.....	190
Cross Settings.....	190
Target Processor.....	193
Standard S32DS C/C++ Compiler.....	195
Standard S32DS C/C++ Linker.....	202
Standard S32DS Assembler.....	205
Standard S32DS Archiver.....	206
Standard S32DS Create Flash Image.....	207

Standard S32DS Create Listing.....	208
Standard S32DS Print Size.....	210
Standard S32DS C/C++ Preprocessor.....	211
Standard S32DS Disassembler.....	211
Folders and files.....	214
Project structure.....	214
Product directory structure.....	216

Part I

Quick Start Guide

Topics:

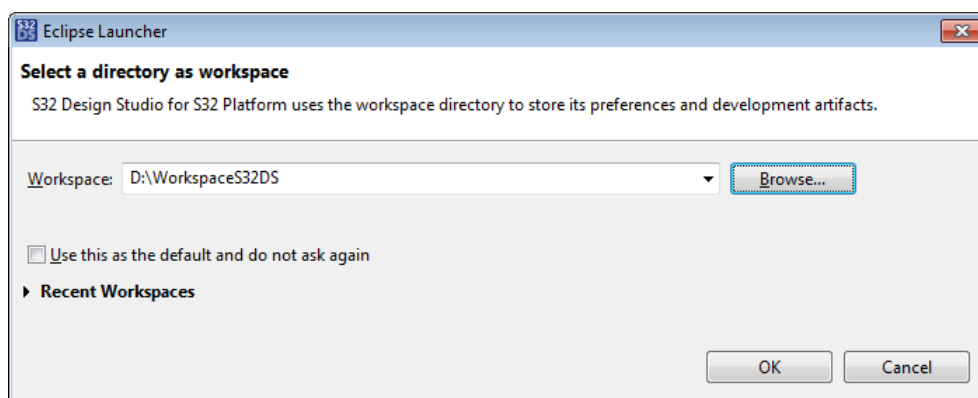
- [Starting S32DS 3.4](#)
- [Creating and building a project](#)
- [Debugging a project](#)
- [Quick links](#)

Starting S32DS 3.4

To start S32 Design Studio for S32 Platform and begin to work with it:

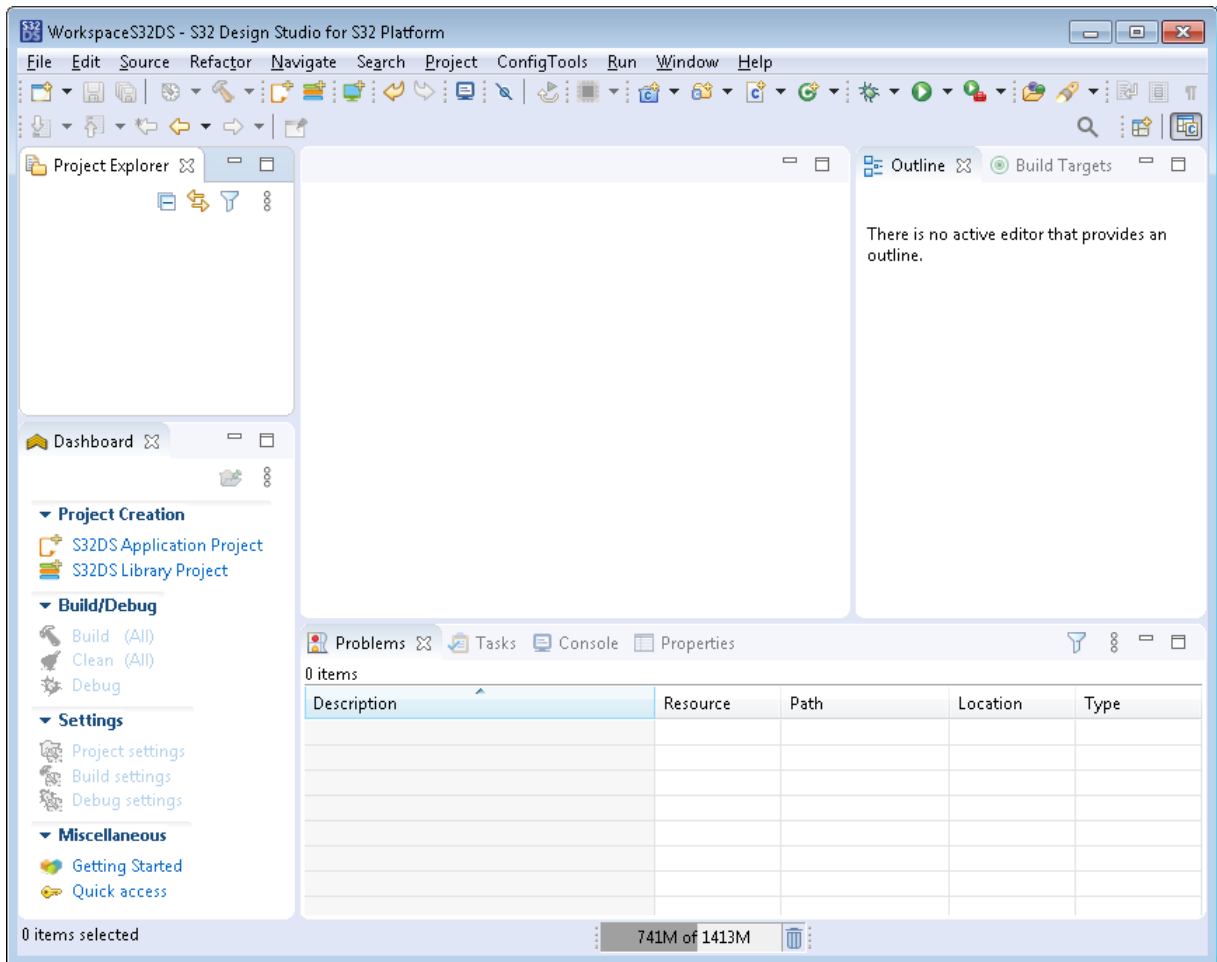
1. Launch S32 Design Studio for S32 Platform: locate the shortcut depending on your selection during the installation, and double-click the product icon.

The **Eclipse Launcher** dialog box appears to let you define the location of your workspace.



Note: A workspace is the folder where S32 Design Studio for S32 Platform stores projects that you create or import.

2. Select a folder for your workspace. It is recommended to create a new workspace for each product instance.
 - To choose the default location, click **OK**.
 - To use a different location, click **Browse**. In the **Select Workspace Directory** dialog box, select the preferred folder or click **Make New Folder** to create a new folder for storing your projects. Click **OK**.
3. S32 Design Studio for S32 Platform is launched. Browse through the **Getting Started** tab and close it. The workbench appears:



Creating and building a project


To create and build a project:

1. Click **File > New > S32DS Application Project** or **S32DS Library Project** on the menu bar. The first page of the wizard appears.
2. Specify the name of the new project in the **Project name** text box.

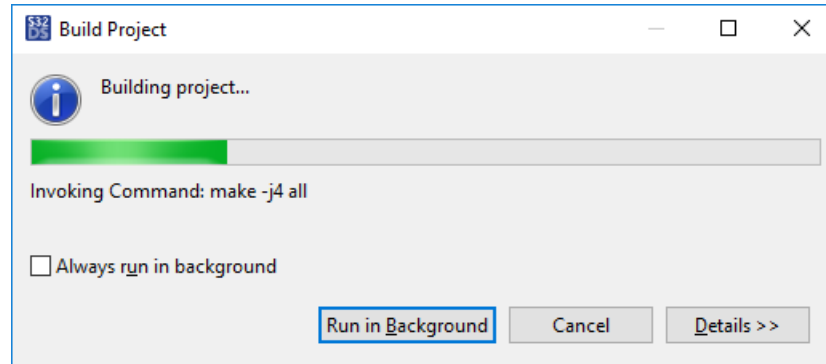
Note: The **Location** field shows the default project location. If you want to change this location, clear the **Use default location** check box, click **Browse** and specify a different location. Click **OK**.

3. Select the target processor from the **Processors** panel.
4. Click **Next**. The second page of the wizard appears.
5. Check the project settings, select the cores and parameters. Click **Finish**.
The new project appears in the **Project Explorer** view.

Note: The wizard creates one or multiple projects, depending on the number of selected cores.

6. To build your project, do any of the following:
 - Right-click the project in the **Project Explorer** view and click **Build Project**
 - Select the project in the **Project Explorer** view, then click **Project > Build Project** on the menu
 - Select the project in the **Project Explorer** view and click  on the toolbar


The build process starts.

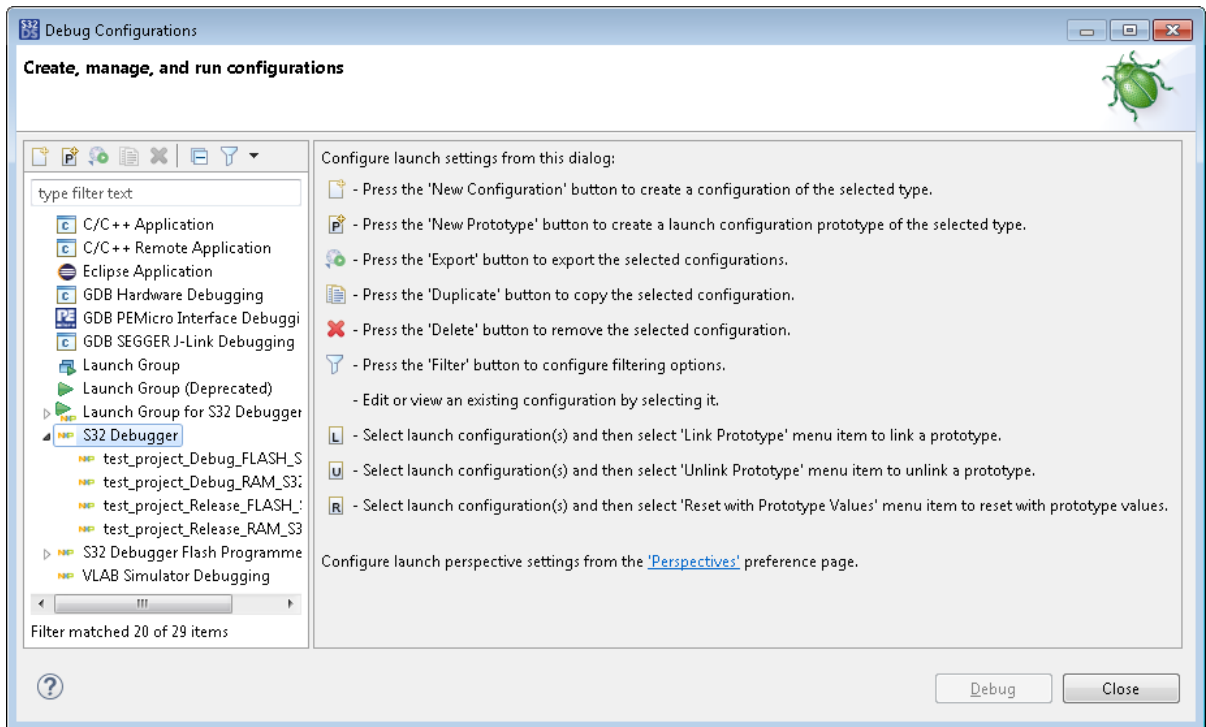


7. If a build generates any errors or warnings, you can see them in the **Problems** view. Read through the build messages in the **Console** view to make sure that the project is built successfully.

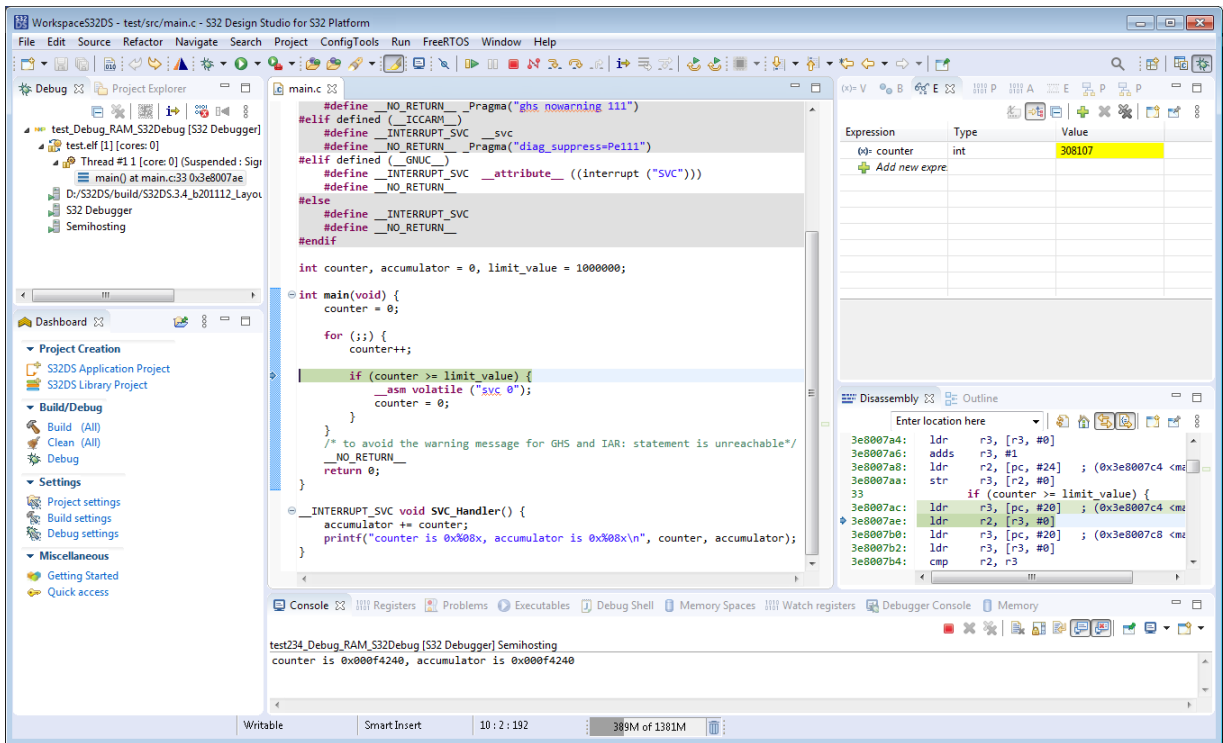
Debugging a project

To debug a project:

1. Set the debug configuration for your project.
 - a) Select the project in the **Project Explorer** view.
 - b) Open the debug configuration in any of these ways:
 - Right-click the project and select **Debug as > Debug Configurations...** from the context menu.
 - Choose **Run > Debug Configurations...** from the menu bar.
 - Click an arrow next to  on the toolbar and select **Debug Configurations...**
 - c) In the **Debug Configurations** dialog box, select the required debug configuration. The name of the debug configuration is composed of the project name, debugging interface and build configuration. The configuration settings appear on the tabs.



- d) Modify the configuration settings where required and click **Apply** to save the changes.
2. Click **Debug**. The debugger downloads the program to the memory of the target processor. The **Debug** perspective is displayed. The execution halts at the first statement of the **main()** function. The program counter icon on the marker bar points the next statement to be executed.



3. To set and run to a breakpoint:
 - a) Double-click on the marker bar next to a statement. The breakpoint indicator (blue dot) appears next to the statement.

- b) From the **Debug** view, select **Run > Resume** on the menu bar. The debugger executes all statements up to (but not including) the breakpoint statement.
4. To control the program:
 - a) From the **Debug** view, select **Run > Step Over** from the menu bar. The debugger executes the breakpoint statement and halts at the next statement.
 - b) From the **Debug** view, select **Run > Resume** from the menu bar. The debugger resumes the program execution.
 - c) From the **Debug** view, select **Run > Terminate**. The debug session ends.

Quick links

- S32 Design Studio page (overview, downloads) <https://www.nxp.com/S32DS>
- S32 Design Studio community (for publicly shared cases) <https://community.nxp.com/community/s32/s32ds>
- Technical support (for confidential issues) <https://www.nxp.com/support>

Part II

Tasks

Topics:

- [Installation management](#)
- [License management](#)
- [Project management](#)
- [Building projects](#)
- [Debugging](#)
- [SDK management](#)
- [Migration guide](#)
- [Troubleshooting](#)

Installation management

Overview

S32 Design Studio for S32 Platform has a package based structure and is initially installed with the base packages that provide the minimum of libraries and tools.

Base packages

- The S32 Design Studio Platform package provides the basic functionality of the product such as Eclipse bundles and integration mechanisms.
- The S32 Design Studio Platform Tools package includes the basic libraries and tools required by all supported devices.
- The NXP GCC for Arm Embedded Processors package includes the GNU tools.

With the basic installation, the user can launch S32 Design Studio for S32 Platform, open all perspectives and views, create and configure workspaces, and open and build device-specific projects that were imported or created earlier. New projects for devices cannot be created, and the compiled code cannot be debugged.

Development packages

To support software design for a given family of devices in S32 Design Studio for S32 Platform, you need to install the respective development package. Each new package brings its libraries, tools, SDKs, project examples, and documentation. Once the required package is installed to S32 Design Studio for S32 Platform, you get the tools for creating application projects and library projects for the newly supported devices, and you gain the missing resources for debugging the device-specific code.

Extension and Add-On packages

To further extend support for the selected device, you can additionally install the extension or add-on package with the accelerator support. Each extension package brings its SDKs (or separate SDK package), compiler and assembler tools, or even visual programming tools for you to enhance your solution with code to be executed on the accelerator processor module.

S32DS Extensions and Updates tool

Installing packages and updates to S32 Design Studio for S32 Platform is performed by means of the **S32DS Extensions and Updates** tool. This wizard communicates with the product's website and displays the most actual information about updates and packages available for installation. Additionally, the user is notified about the latest installation candidates with a pop-up box displayed on the desktop.

The user can pick the required updates and packages in the wizard and have them installed in S32 Design Studio for S32 Platform with a couple of clicks. Besides, the wizard displays all installed packages and enables uninstalling of any selected package.

Plug-in products

In addition to device support, the functionality of S32 Design Studio for S32 Platform can be extended with support for new debugging instruments such as hardware debugging interfaces and simulators. Integration of third-party products with S32 Design Studio for S32 Platform is implemented through plug-ins.

Though the **S32DS Extensions and Updates** wizard can be configured for software lookup on any site, installation of plug-ins into S32 Design Studio for S32 Platform using this tool is not supported. To install a plug-in, use the functionality provided by the Eclipse platform.

Getting product updates

S32 Design Studio for S32 Platform patches and service packs come as updates. S32 Design Studio for S32 Platform implements the logic to find updates on the specified sites, load them to your computer, and notify you about updates available for installation. To be in sync with the latest updates available for your product, do the following:

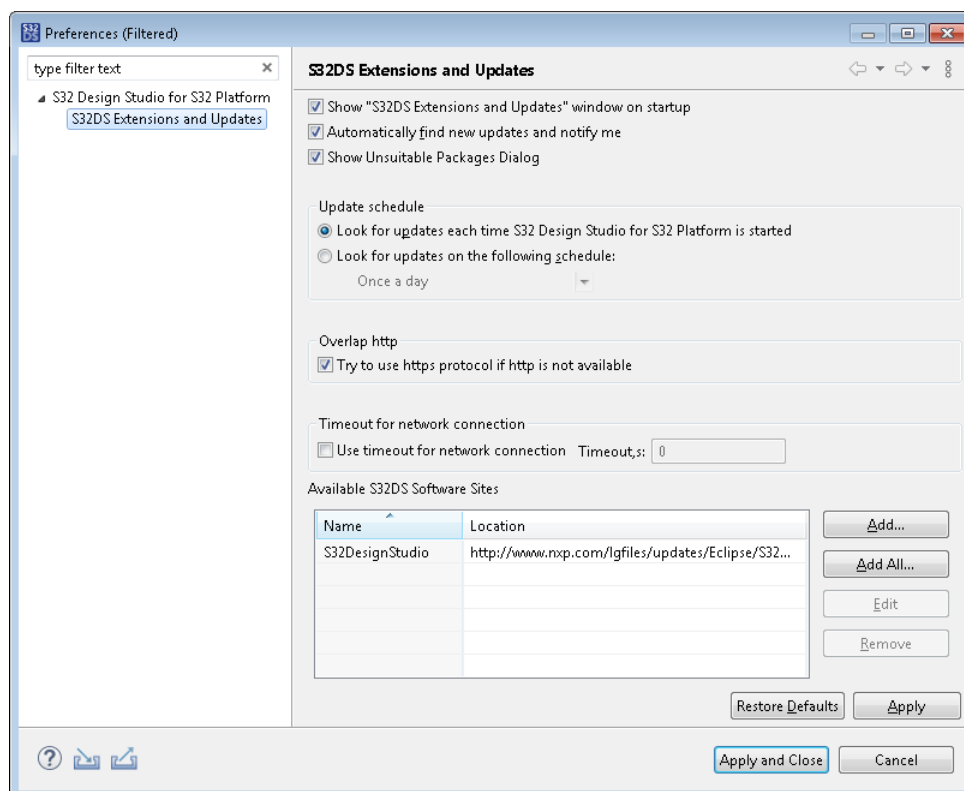
1. Create the list of sites where S32 Design Studio for S32 Platform would look for updates. Find the details in topic [Managing software sites](#).
2. Configure S32 Design Studio for S32 Platform to look for the latest updates automatically. Find the details in topic [Getting updates automatically](#).
3. Install the latest updates when notified. Find the details in topic [Installing product updates and packages](#).

In addition, you can download the latest product updates on your computer from the website manually. Find the details in topic [Downloading updates manually](#).

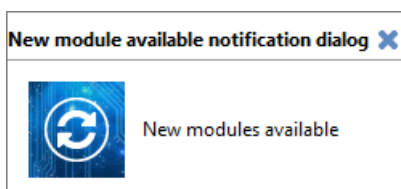
Getting updates automatically

S32 Design Studio for S32 Platform can be configured to look for updates automatically at every startup or by schedule. The search is performed across the sites that you need to specify in the product preferences. Find the details in topic [Managing software sites](#).

To enable automatic lookup for updates, click **Window > Preferences** on the menu and go to **S32 Design Studio for S32 Platform > S32DS Extensions and Updates**. To activate automatic lookup for updates, select the **Automatically find new updates and notify me** option. Select the preferred time for lookup and click **Apply**.



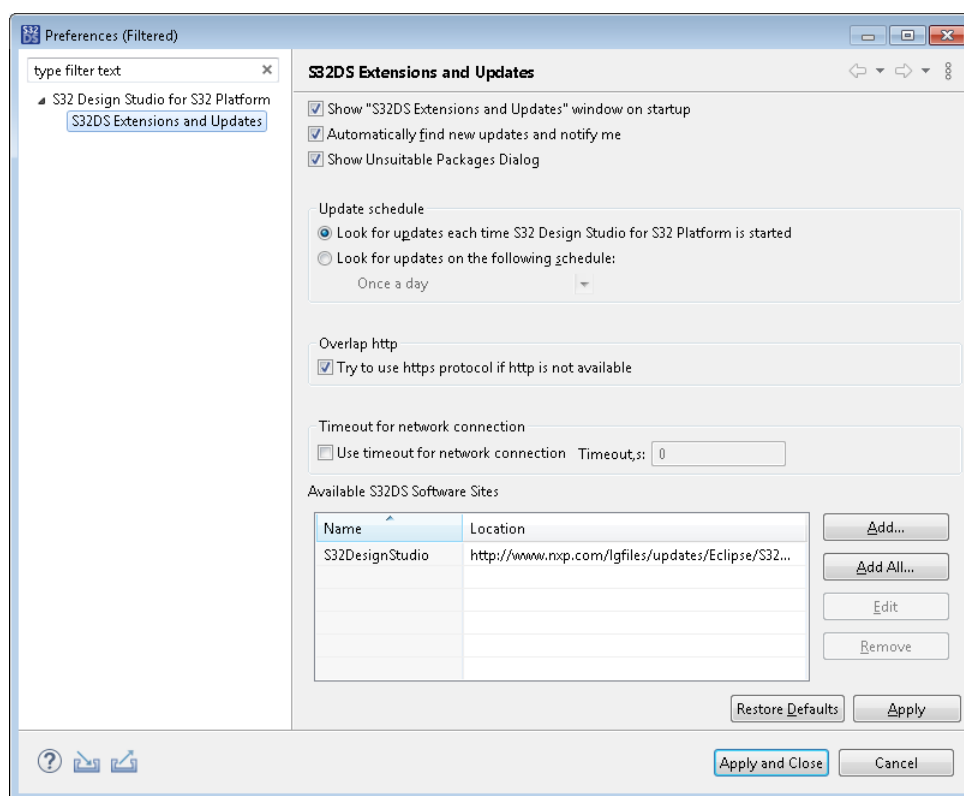
When automatic lookup finds available updates, you get a notification:



Click the notification box to launch the **S32DS Extensions and Updates** wizard. Install the update as described in topic [Installing product updates and packages](#).

Managing software sites

S32 Design Studio for S32 Platform looks for updates and software packages across the specified software sites. To view and edit these sites, click **Manage Sites** in the **S32DS Extensions and Updates** wizard. Alternatively, you can click **Window > Preferences** on the main menu and go to **S32 Design Studio for S32 Platform > S32DS Extensions and Updates** on the left pane of the **Preferences** window.



The right pane of the **Preferences** window displays all registered software sites added by default.

To manage the list of software sites, use the buttons located at the right side of the window.

Table 1: Managing software sites

Purpose	Action
Add a software site to the list	<ol style="list-style-type: none"> Click Add. In the Add Site dialog box, specify the location of the site: <ul style="list-style-type: none"> To add a local or network folder for the lookup, click Local. Browse to the required folder and click OK.

Purpose	Action
	<ul style="list-style-type: none"> To add an archived file (JAR or ZIP) for the lookup, click Archive. Browse to the required file (local or network) and click Open. To add a web page for the lookup, type the required URL to the Location field. <p>3. Specify the name of the new site and click OK.</p>
Add several archived files	<p>1. Click Add All.</p> <p>2. Browse to the required files (local or network), select them and click Open.</p>
Edit the name or location of the site	Click Edit and type the preferred name and location in the dialog box. Click OK .
Permanently remove a site from the list	Select the site in the list and click Remove .

Click **Apply** before closing the **Preferences** dialog box for the latest updates to take effect.

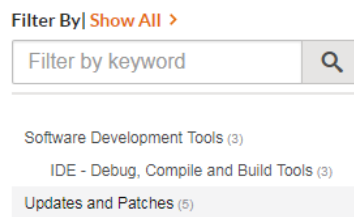
Note: If you added a software site or archive file incompatible with the current product version, you get the "Unsuitable Package for Platform" notification in the **S32 Extensions and Updates** wizard, and new software does not appear in the list of available packages. If you don't want to see these notifications, disable the **Show Unsuitable Package Dialog** option at the top of the Preferences window.

Downloading updates manually

You can download updates and patches for S32 Design Studio for S32 Platform directly from the NXP website and install them offline.

To download updates from the site:

- Go to the nxp.com/S32DS web page and click S32 Design Studio for S32 Platform.
- On the product page, go to the **Downloads** tab and find the **Updates and Patches** section on the menu (if available).



- To download an update or a patch, click **Download** and save the archive file in a local folder.

To install the downloaded software, add the archive file to the list of lookup sites as described in topic [Managing software sites](#). After that, the update becomes available in the **S32DS Extensions and Updates** wizard.

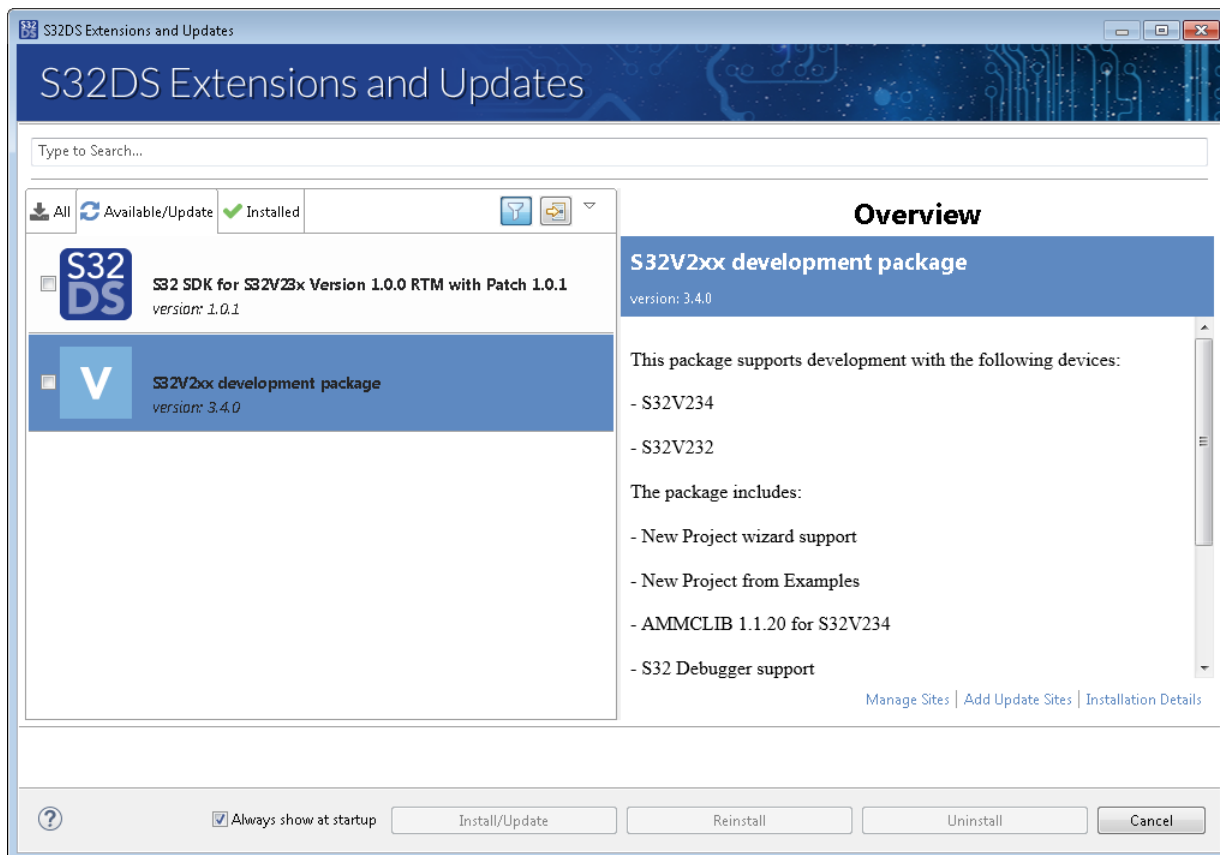
Installing product updates and packages

S32 Design Studio for S32 Platform provides a tool to help you find and install the latest product updates and optional software packages. The lookup is performed across the sites that are specified in the product preferences. To learn how to preview and edit these sites, refer to topic [Managing software sites](#).

To install updates and additional packages to S32 Design Studio for S32 Platform:

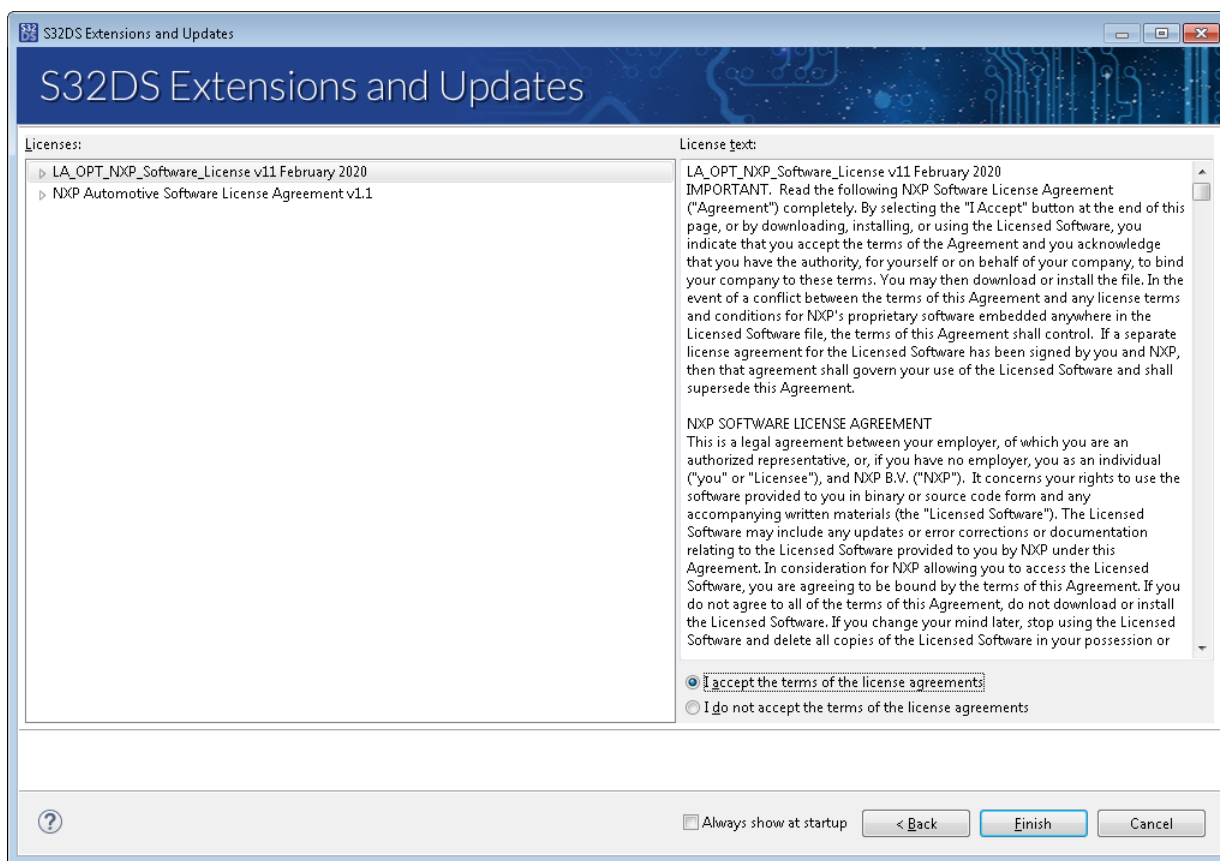
- From the menu, click **Help > S32DS Extensions and Updates**.

2. In the left pane of the **S32DS Extensions and Updates** wizard, find all software packages already installed and ready to be installed.
 - Use the **All**, **Available/Update** and **Installed** tabs to show and hide packages of a respective type.
 - Use the filter buttons to show packages of the required hardware type or category. Removing the filter clears the package selection.



Click a package in the left pane. The right pane loads the description of the package.

3. Check the box on each package that you need to install or update. Click **Install/Update**.
4. On the next page, verify the selected packages and click **Next**.
5. Accept the license terms. Click **Finish**.



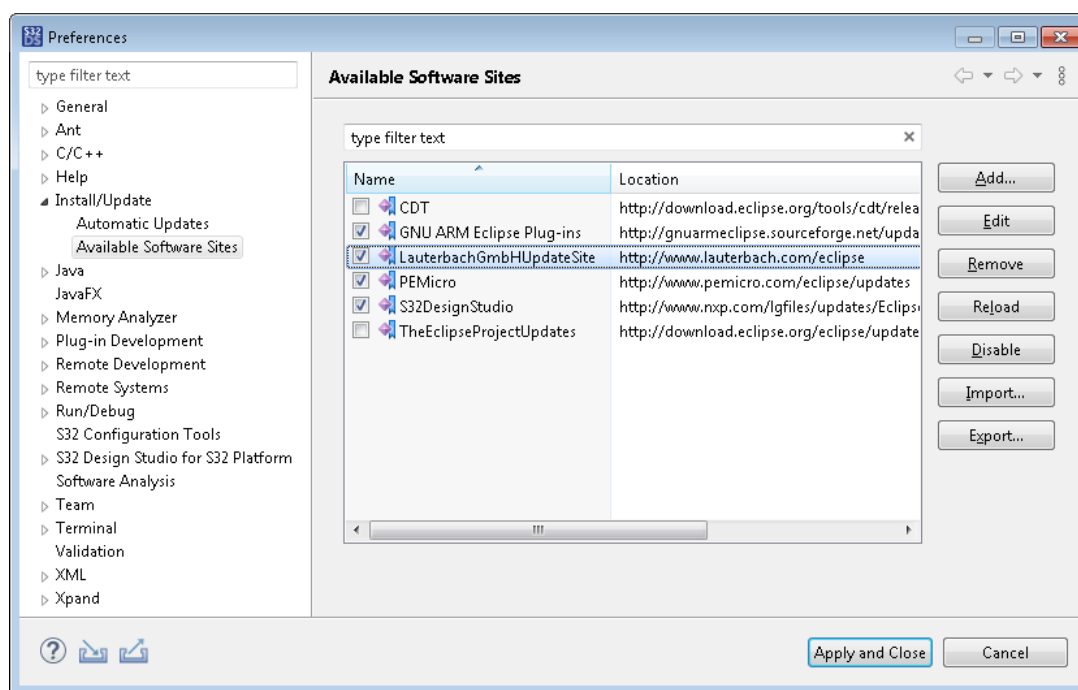
6. After the installation is complete, restart S32 Design Studio for S32 Platform.

Installing plug-ins

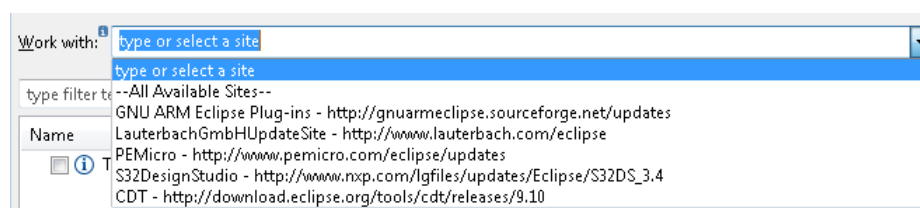
To perform particular tasks, you may need to install plug-ins of third-party software vendors on S32 Design Studio for S32 Platform. For example, to support Lauterbach hardware debug interfaces, S32 Design Studio for S32 Platform requires special plug-ins installed from the manufacturer website.

To install new software to S32 Design Studio for S32 Platform:

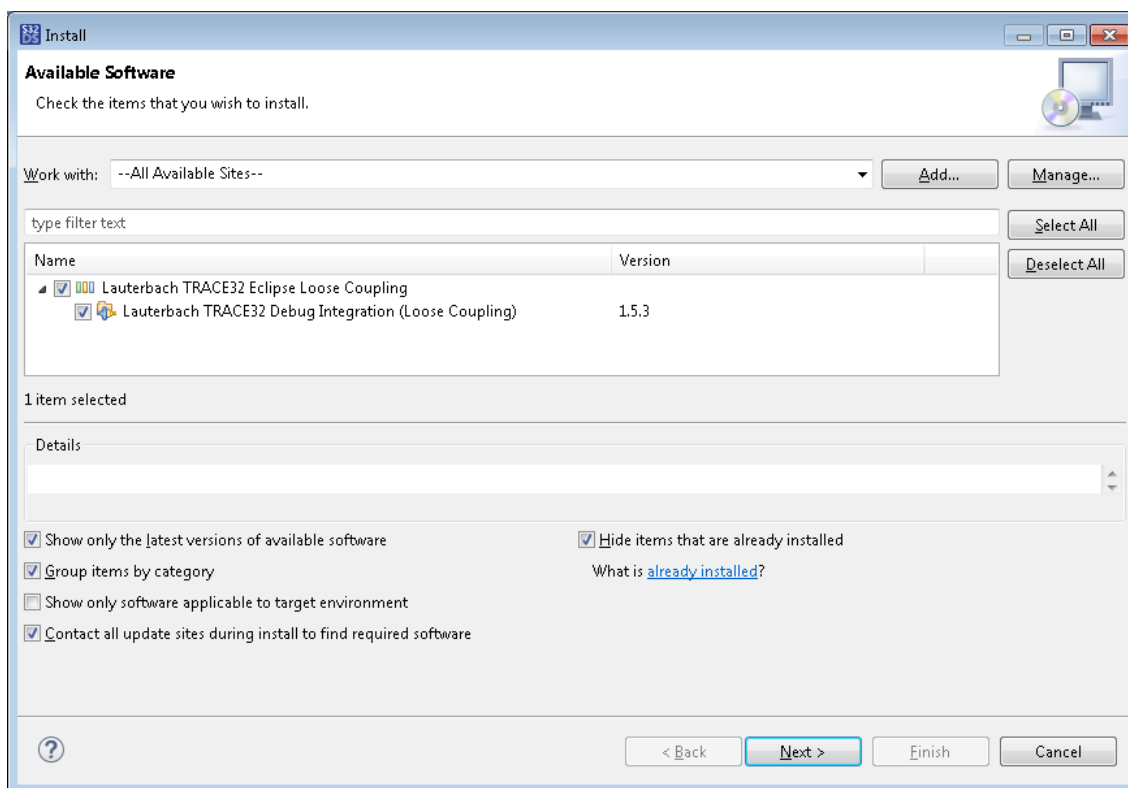
1. On the main menu, click **Help > Install New Software**.
2. In the **Install** wizard, open the **Work with** list and check if the required website is available.
3. If the required site is not in the **Work with** list, click **Manage**. On page **Available software sites** of the **Preferences** dialog box, select the required sites and click **Apply and Close**.



4. In the **Work with** list, select the required site, or select **All available sites** to install new software from all sites.



5. The plug-ins of the selected vendors appear in the list below. Select the software components to be installed. Click **Next** and again **Next**.

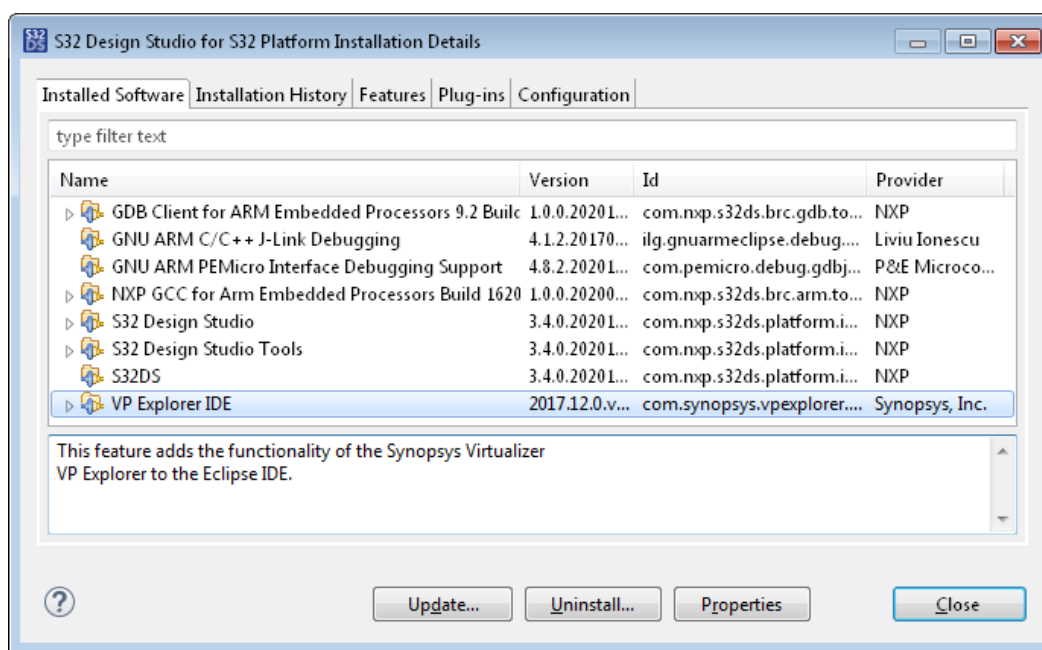


6. On the next wizard page, accept the license terms and click **Finish**.

The selected software is now installed. If prompted, restart S32 Design Studio for S32 Platform to make the new functionality available in dialog boxes and menus of S32 Design Studio for S32 Platform.

Viewing all installed software

The **Installed Software** tab of the **S32 Design Studio for S32 Platform Installation Details** dialog box displays the S32 Design Studio for S32 Platform installation and all installed plug-ins and software packages.



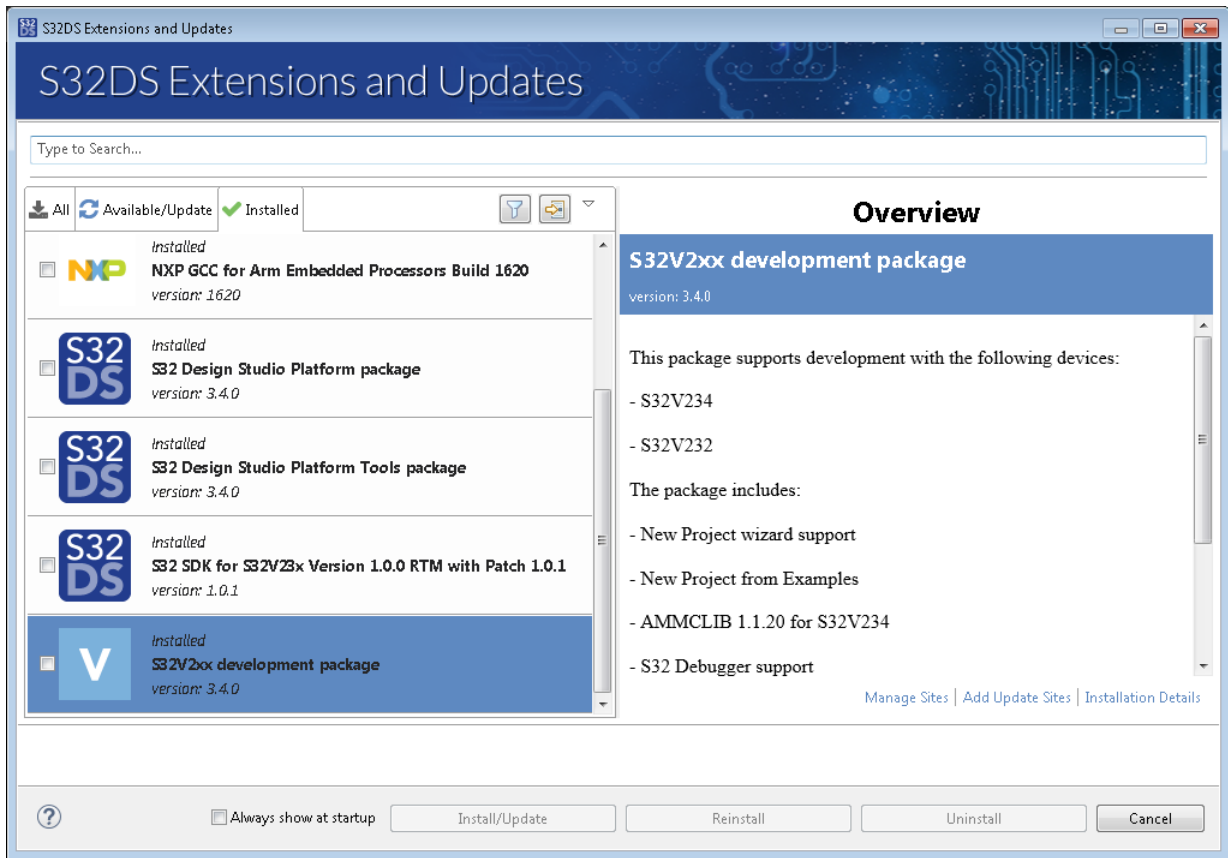
To view the list of plug-ins and software packages installed on S32 Design Studio for S32 Platform, click **Help > Installation Details** on the main menu.

To manage a selected installation, use the buttons located at the bottom of the **S32 Design Studio for S32 Platform Installation Details** dialog box:

- **Update:** Click to apply updates (if any found on the lookup sites) to the selected installation.
- **Uninstall:** Click to uninstall the selected installation. Find the details in topic [Uninstalling plug-ins](#).
- **Properties:** Click to view the summary and the license information for the selected installation.

Viewing installed updates and packages

The **S32DS Extensions and Updates** wizard displays all S32 Design Studio for S32 Platform updates and software packages already installed and ready to be installed.



To view the list of installed product updates and packages:

1. On the main menu, click **Help > S32DS Extensions and Updates**.
2. In the **S32DS Extensions and Updates** wizard, click **Installed** for the left pane to display the installed software only and hide all other kinds of software.

Alternatively, you can click **Installation details** to view a list of installed software. The list includes each package's name and version. Click **Copy to Clipboard** to share your configuration.

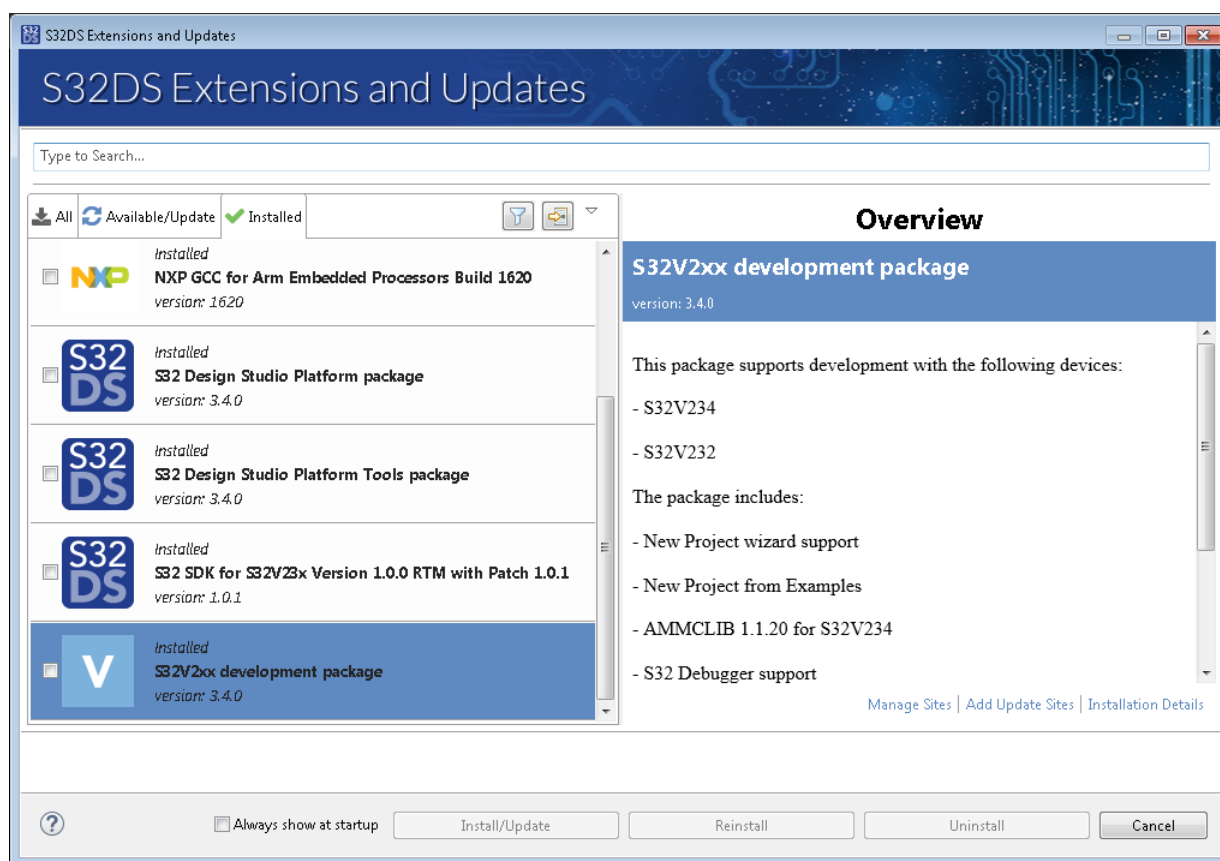
Get more information about the installed packages on page **Getting Started** of the S32 Design Studio for S32 Platform. To open this page, click **Help > Getting Started** from the menu.

The **Extensions** tab displays all recently installed packages. Each package appears on the tab as a box with the **More** button. Click it to open the overview page with links to all available documentation.

Uninstalling packages

To uninstall a software package:

1. On the main menu, click **Help > S32DS Extensions and Updates**.
2. In the **S32DS Extensions and Updates** wizard, click **Installed** for the left pane to display the installed software only and hide all other kinds of software.

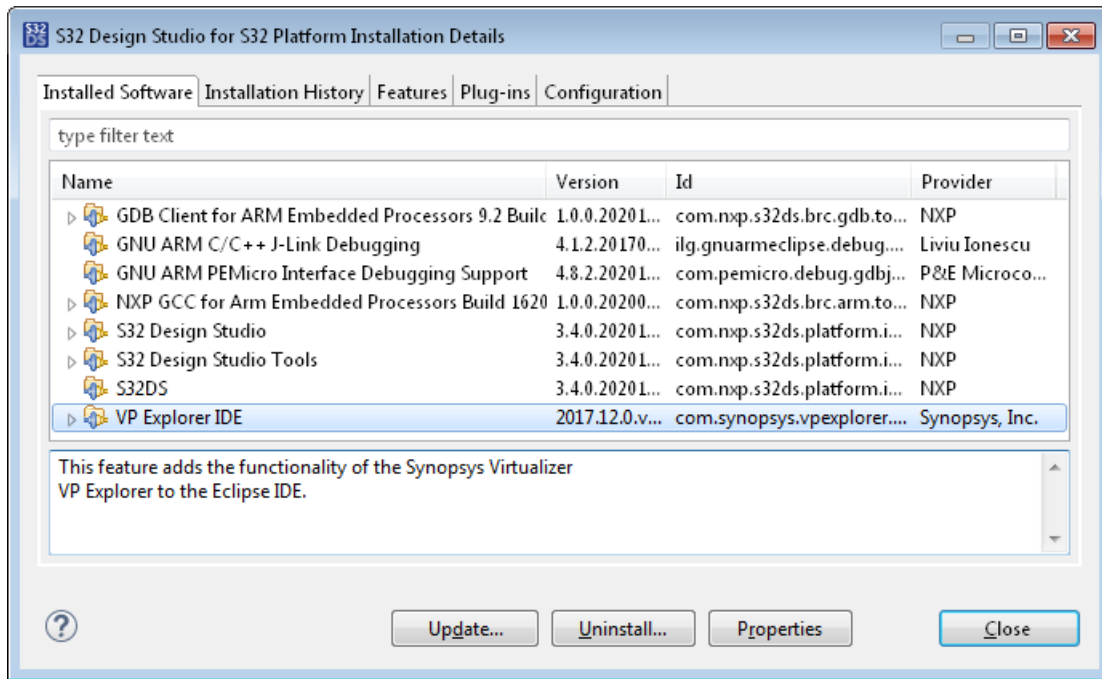


3. Check the box on each package that you need to uninstall. Click **Uninstall**.
4. After the uninstall operation is finished, restart S32 Design Studio for S32 Platform.

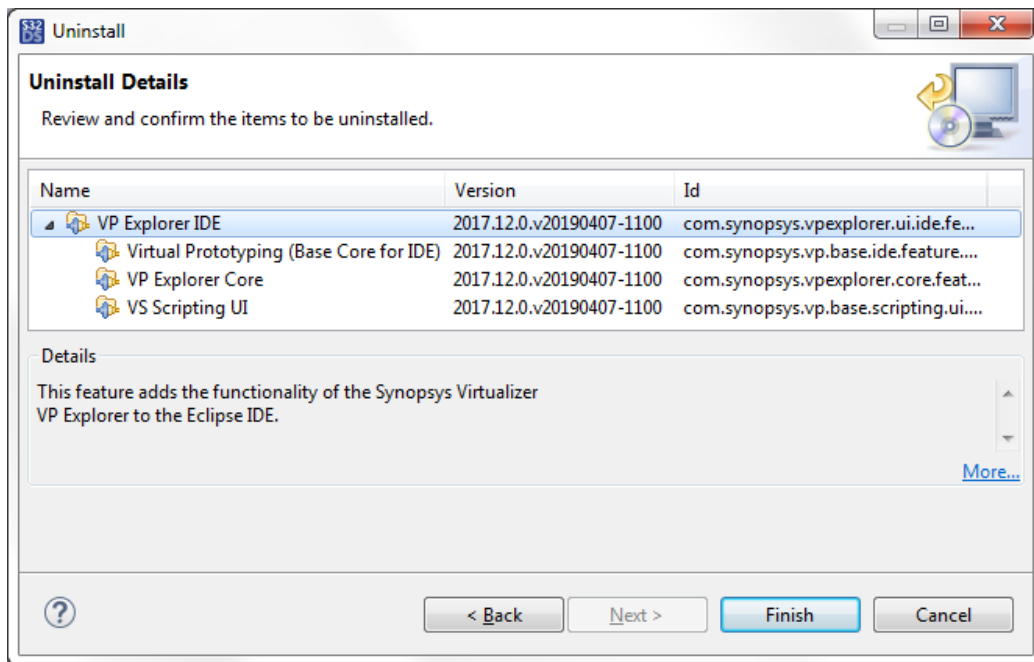
Uninstalling plug-ins

To uninstall a third-party plug-in:

1. On the main menu, click **Help > Installation Details**.
2. In the **S32 Design Studio for S32 Platform Installation Details** dialog box, open the **Installed Software** tab. Select the software to be removed and click **Uninstall**.



3. In the message box, click **Yes** to confirm the review of components to be uninstalled from S32 Design Studio for S32 Platform. Click **Finish** to complete the operation.



4. When the uninstall operation is done, restart S32 Design Studio for S32 Platform.

Uninstalling S32DS 3.4

To uninstall S32 Design Studio for S32 Platform:

1. If you have installed any third-party plug-ins on S32 Design Studio for S32 Platform, uninstall them as described in [Uninstalling plug-ins](#).

2. Close all applications that use resources of S32 Design Studio for S32 Platform.
3. Run the uninstaller:
 - On Windows, go to the product installation directory, open the _S32 Design Studio for S32 Platform 3.4_installation folder, and run the uninstaller. Or, use **Control Panel > Uninstall a program**.
 - On Windows 10, go to **Settings > System > Apps & features > S32 Design Studio 3.4 > Uninstall**.
 - On Linux, open the terminal, go to the directory with the installed product, type `./Uninstall` and press **Enter**.
4. In the wizard, pass through all steps and click **Finish**.
5. When done, the wizard displays the **Uninstall complete** page. Close the wizard.

License management

Overview

S32 Design Studio for S32 Platform runs on basis of a free software license. The product is ready for use once the license is activated in the Flexera license management center. The validity period of the license is four years long.

Getting the activation key

After getting S32 Design Studio for S32 Platform from the product's website, the user receives the activation key to the specified email address. Also, the activation key can be obtained from the user's account on the product website.

License activation

Activation of the license can be done when installing the product. The license can be activated online or offline.

- Online activation means that S32 Design Studio for S32 Platform holds the entire conversation with the license management service over the Internet automatically. The user only enters the activation key when requested.
- Offline activation requires more user effort and serves when online activation cannot be applied, for instance, when the host machine cannot access Internet services. The user generates the activation request and places it to a location that can be accessed from the Internet. Then the user visits the user account on the product's website, uploads the activation request, generates the activation response, and submits it to S32 Design Studio for S32 Platform.

License status

The status of the license (activated or not) can be learnt from the user account on the product's website. The license information is also available in the **Product Licenses** window of S32 Design Studio for S32 Platform.

Returning the license

If you decide to uninstall S32 Design Studio for S32 Platform or reinstall it to a different workstation, the best approach is to return the license and thus keep it for future use. The license can be returned online or offline, similar to the activation procedure.

When reinstalled, S32 Design Studio for S32 Platform can be activated with the returned license. The license can be reused with the same product version only. If the product was upgraded to the higher version, the new license is required.

Licensing S32DS 3.4

To activate the license for your S32 Design Studio for S32 Platform, you need to submit the activation code when installing the product. For the detailed instructions, refer to the following topics:

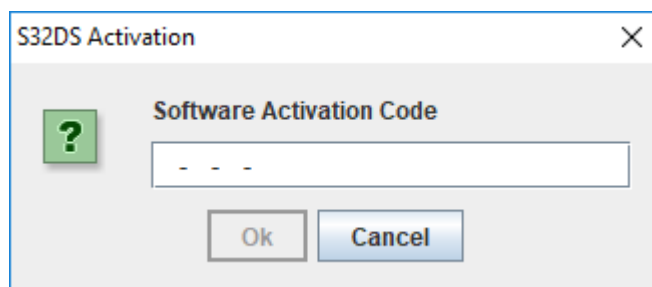
- [Applying the license during installation](#)

- [Getting the activation code](#)

Applying the license during installation

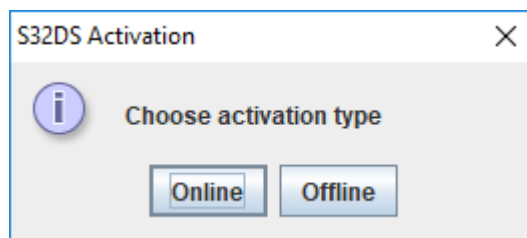
To apply the license when installing S32 Design Studio for S32 Platform:

1. Get the activation code as described in topic [Getting the activation code](#).
2. When requested, enter the activation code and click **OK**.



If you click **Cancel**, the licensing is skipped and the installation continues.

3. After the code is entered, choose the activation type. Click **Online** for the installer to complete the activation without your interference.



Click **Offline** to submit the activation files manually. This option may be useful if you are not connected to the Internet.

4. If you click **Offline**, the installer generates the activation request. Perform the following steps:
 - a) Save the `request.xml` file in a local folder.
 - b) Browse to www.nxp.com and log in with your registered member credentials.
 - c) On the web page, go to **Offline Activation**. Browse to the `request.xml` file that you saved recently and click **Process**. Then save the generated `activation.xml` file in a local folder.

NXP > Software & Support > **Offline Activation**

Software & Support

- Product List
- Product Search
- Order History
- Recent Product Releases
- Recent Updates

Licensing

- License Lists
- Offline Activation**

FAQ

- Download Help
- Table of Contents
- FAQs

Reference your saved request XML file in the form below. After clicking activate, you will be prompted to download the activation response. Save it and load it into your application to activate.

request.xml

- d) Get back to the installer. In the **Activation response** dialog box, browse to the `activation.xml` file and click **Load**.

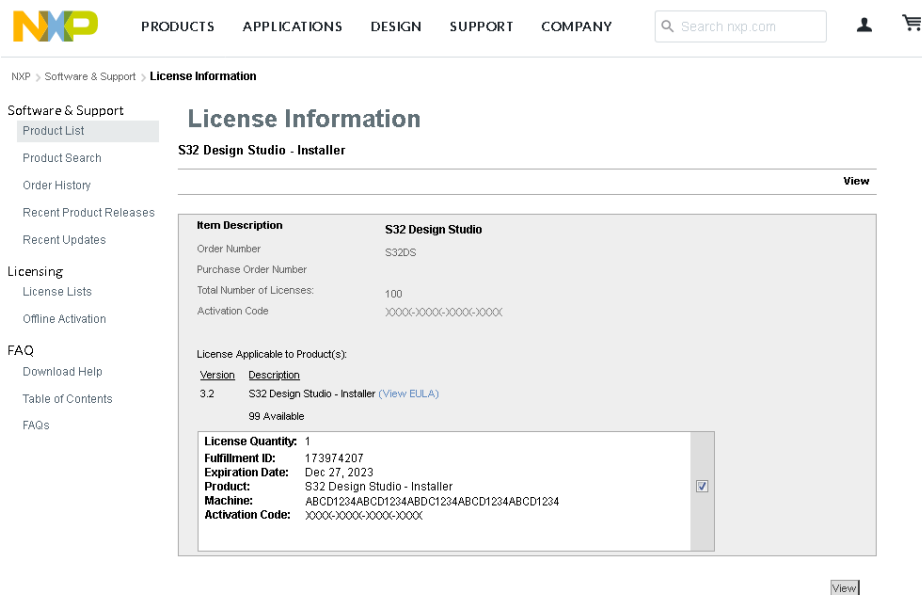
Once done, the activation step is passed and the installation continues.

Getting the activation code

When you agree to the license terms before downloading the S32 Design Studio for S32 Platform installer, you get a message with the activation code to the email address specified in your NXP account. Or, you can get the activation code directly from your account on the website.

To get the activation code on the NXP website:

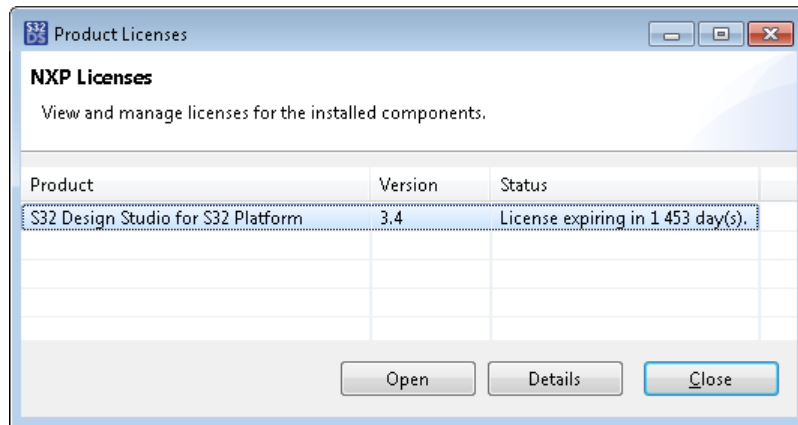
1. Browse to www.nxp.com.
2. Click **Account** and log in to the site using your registered member ID and password.
3. Click your login name on top of the page, then click **My account** and **Software Licensing and Support**.
4. Click **Product List** under **Software&Support**. Click your product in the list.
5. Click **I Agree** to accept the terms and conditions.
6. On the **Product Download** page, go to **License Keys**. The license information for your product includes the activation code.



Viewing the license information

To view the license information for S32 Design Studio for S32 Platform:

1. On the main menu, go to **Help > NXP Licenses**.
2. In the **Product Licenses** dialog box, find the list of licenses related to the product.



3. Select the product and click **Details** to see more information about the license.

Viewing licenses on the website

Your NXP account keeps the information about all requested product licenses. To view this information:

1. Browse to www.nxp.com.
2. Click **Account** and log in to the site using your registered member ID and password.
3. Click your login name on top of the page, then click **My account** and **Software Licensing and Support**.
4. To see all products for which you have a license, click **Product List** under **Software&Support**.

To view the detailed license information, click the product and go to the **License Keys** tab.

The screenshot shows the NXP website's 'License Information' page for 'S32 Design Studio - Installer'. The page includes a navigation menu with 'PRODUCTS', 'APPLICATIONS', 'DESIGN', 'SUPPORT', and 'COMPANY'. A search bar is present with the text 'Search nxp.com'. The main content area is titled 'License Information' and 'S32 Design Studio - Installer'. It displays a table with the following information:

Item Description	S32 Design Studio
Order Number	S32DS
Purchase Order Number	
Total Number of Licenses:	100
Activation Code	XXXXXX-XXXXXX-XXXXXX

Below the table, it states 'License Applicable to Product(s):' and provides a table with columns 'Version' and 'Description':

Version	Description
3.2	S32 Design Studio - Installer (View EULA)

It also indicates '99 Available' and shows a 'License Quantity: 1'. A detailed license information box is visible, containing:

- License Quantity: 1
- Fulfillment ID: 173974207
- Expiration Date: Dec 27, 2023
- Product: S32 Design Studio - Installer
- Machine: ABCD1234ABCD1234ABCD1234ABCD1234ABCD1234
- Activation Code: XXXXX-XXXXXX-XXXXXX-XXXXXX

5. To see all activated licenses, click **License Lists** under **Licensing**. To view the detailed license information for a product, click **Additional Details** or **Details**.

The screenshot shows the NXP website's 'Search Licenses' page. The navigation menu includes 'Products', 'Applications', 'Design', 'Support', and 'About'. A search bar is present with the text 'Search nxp.com'. The main content area is titled 'Search Licenses' and includes a 'Find Licenses' section with a search dropdown and a 'Filter By' section with radio buttons for 'Order Date', 'Entitlement Exp. Date', and 'License Expiration Date', along with 'From' and 'To' date input fields. A 'Search Licenses' button is located below the filters.

Below the search area, there is a table with the following columns: Fulfillment ID, Details, License Exp., Entitlement Exp., and Action. The table contains two rows of license information:

Fulfillment ID	Details	License Exp.	Entitlement Exp.	Action
144244844	Catalog Item Name S32 Design Studio for Power Architecture Additional Details	Jun 3, 2023	Jun 3, 2023	Return Details
559557055	Catalog Item Name LAX Compiler Additional Details	Mar 21, 2023	Mar 21, 2023	Return Details

Returning the license

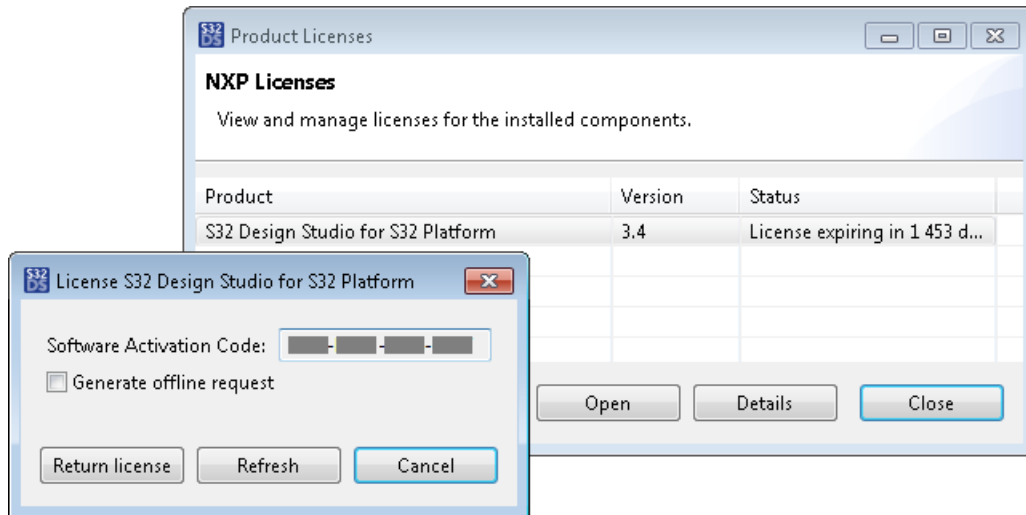
Before reinstalling or migrating your S32 Design Studio for S32 Platform 3.4 to a different computer, you need to return the license. Later, you can activate this license again for a different installation of the product.

To return the license:

1. Go to www.nxp.com.
2. Click **Account** and log in to the site using your registered member ID and password.

3. Click your login name on top of the page, then click **My account** and **Software Licensing and Support**.
4. To see all activated licenses, click **License Lists** under **Licensing**.
5. In the product entry, click **Return**.

Another option to return the license is available from the **Help > NXP Licences** menu of S32 Design Studio for S32 Platform 3.4.



In the **Product Licenses** dialog box, click **Open**.

- To return the license online, click **Return license** in the popup dialog box.
- When not connected to the Internet, return the license offline. Select the **Generate offline request** option and click **Return license**. Save the deactivation request (`request.xml`) and submit it in your NXP account as described in topic [Applying the license during installation](#). Save the activation file to be able to activate the license in the future.

Now the S32 Design Studio for S32 Platform 3.4 license is excluded from the list of active licenses in your NXP account (**Licensing > License Lists**).

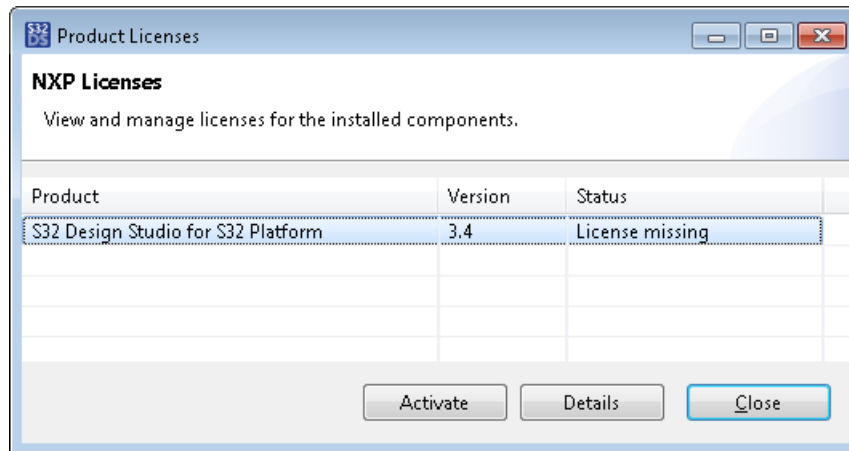
Note: When you return a license, it affects all component and product instances that depend on this license.

Relicensing S32DS 3.4

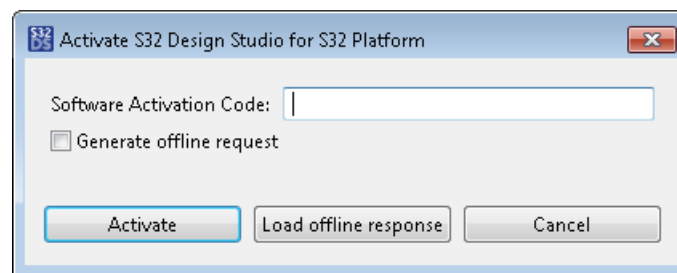
If the product license was returned by mistake or with the purpose of using a different license, activate the license as described below.

To relicense the product:

1. Launch S32 Design Studio for S32 Platform 3.4.
2. In the **Product Licenses** dialog box, the product entry appears with the "License missing" status. Click **Activate**.



3. In the **Activate S32 Design Studio for S32 Platform 3.4** dialog box, do the following:



- For online activation, enter the activation code and click **Activate**. Learn how to get the activation code in topic [Getting the activation code](#).
- When not connected to the Internet, activate the license offline. Select the **Generate offline request** option and click **Activate**. Then submit the generated `request.xml` file and get the `activation.xml` response as described in topic [Applying the license during installation](#). Click **Load offline response**, browse to the `activation.xml` file and click **Load**.

Once the license is activated, the **Product Licenses** dialog box displays the number of days before expiration in the **Status** field.

Project management

Overview

In S32 Design Studio for S32 Platform, you start a new project for every embedded application or a library you are about to design. A project serves as a container, keeping the source code, configuration files, and resources of your program arranged in folders within your workspace.

Starting a project

There are several ways how you can start a project in S32 Design Studio for S32 Platform. To design a project from scratch, use a project creation wizard. The wizard guides you through a series of steps, helping you to name the project, to specify the target hardware and to select the appropriate programming language (C or C++), toolchains, SDKs, and the debugger.

The other ways to start a project are importing an existing project to your workspace and reusing a project example, either installed with S32 Design Studio for S32 Platform or created manually.

Project folders and files

Once created or copied to the workspace, a project is stored in the project's root folder named similarly. The project files inside the root folder are arranged in the hierarchy of folders with the predefined names and locations. You can extend your project with custom folders and files added under the root folder whenever necessary. In addition, a project can link files and folders located outside the projects' root folder.

Device configuration

Projects for particular processor families can include device configuration files. To design a device configuration, you create a project with enabled support for the S32 Configuration tool. The default device configuration is generated within the project and can be edited in special perspectives, including in the graphics mode. All updates are transformed to the XML format and code and saved to the configuration files.

Project properties

Every project includes a collection of properties that configure the build process. You can open and edit the project properties in a dialog box. This dialog is the place where you specify the project variables, choose optional builders, update the build configuration settings and do adjustments in the selected toolchain, reference other projects and SDKs, and create and edit launch configurations.

Visibility and operations

At design time, all operations with projects are performed in the **Project Explorer** view. The **Project Explorer** shows the file and folder structure of all opened projects available in the workspace. The context menu called on the project's root folder or on its nested folder enables you to perform the following actions:

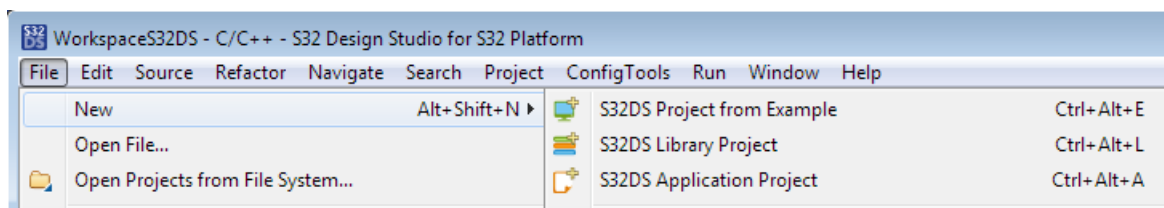
- Opening and closing projects
- Adding and linking files and folders to projects
- Building projects or paths
- Exporting projects or particular folders
- Removing projects or particular folders from the workspace
- Accessing the project or folder properties

The topics included in this chapter provide more information on how to manage projects in S32 Design Studio for S32 Platform.

Starting a project

Creating a project in the wizard

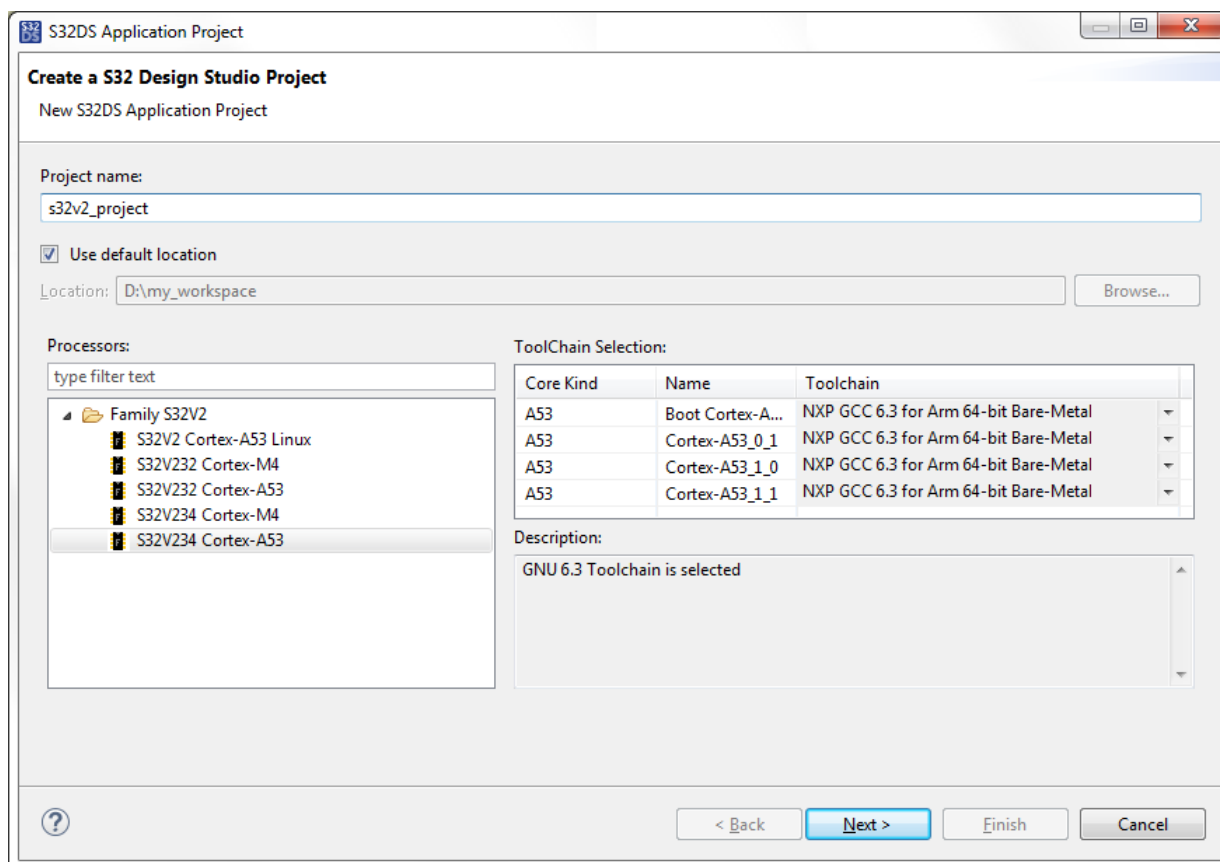
To develop an embedded application or a library, you generally start with creating a new project in the project creation wizard. There are several project creation wizards available from the **File > New** menu:



- **S32DS Application Project:** Creates an application project for a certain processor, core, language, and toolchain.
- **S32DS Library Project:** Creates a library project for a certain processor, core, language, and toolchain.
- **S32DS Project from Example:** Helps you quickly create an application project that copies all source files and project settings from a project example. This option is described in topic [Creating a project from an example](#).

The high-level steps for creating a project are as follows:

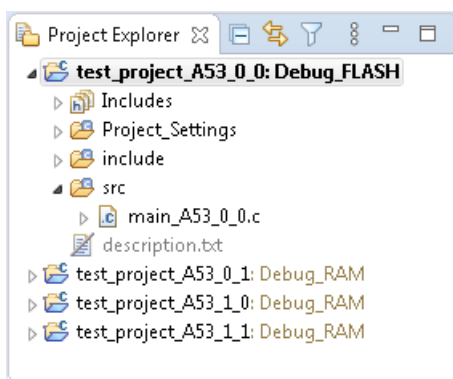
1. Launch the project creation wizard for your type of project.
2. On the first page of the wizard, specify the project name and location, the processor, and the toolchain for each core.



3. Click **Next**.
4. On the second page of the wizard, select the required cores and specify the settings for each core.
5. Click **Finish**.

Note: Learn about all wizard fields in the **Project creation wizards** section of this documentation (**Reference > User interface > Wizards**).

The wizard creates all project folders and files and displays the new project in the **Project Explorer**:

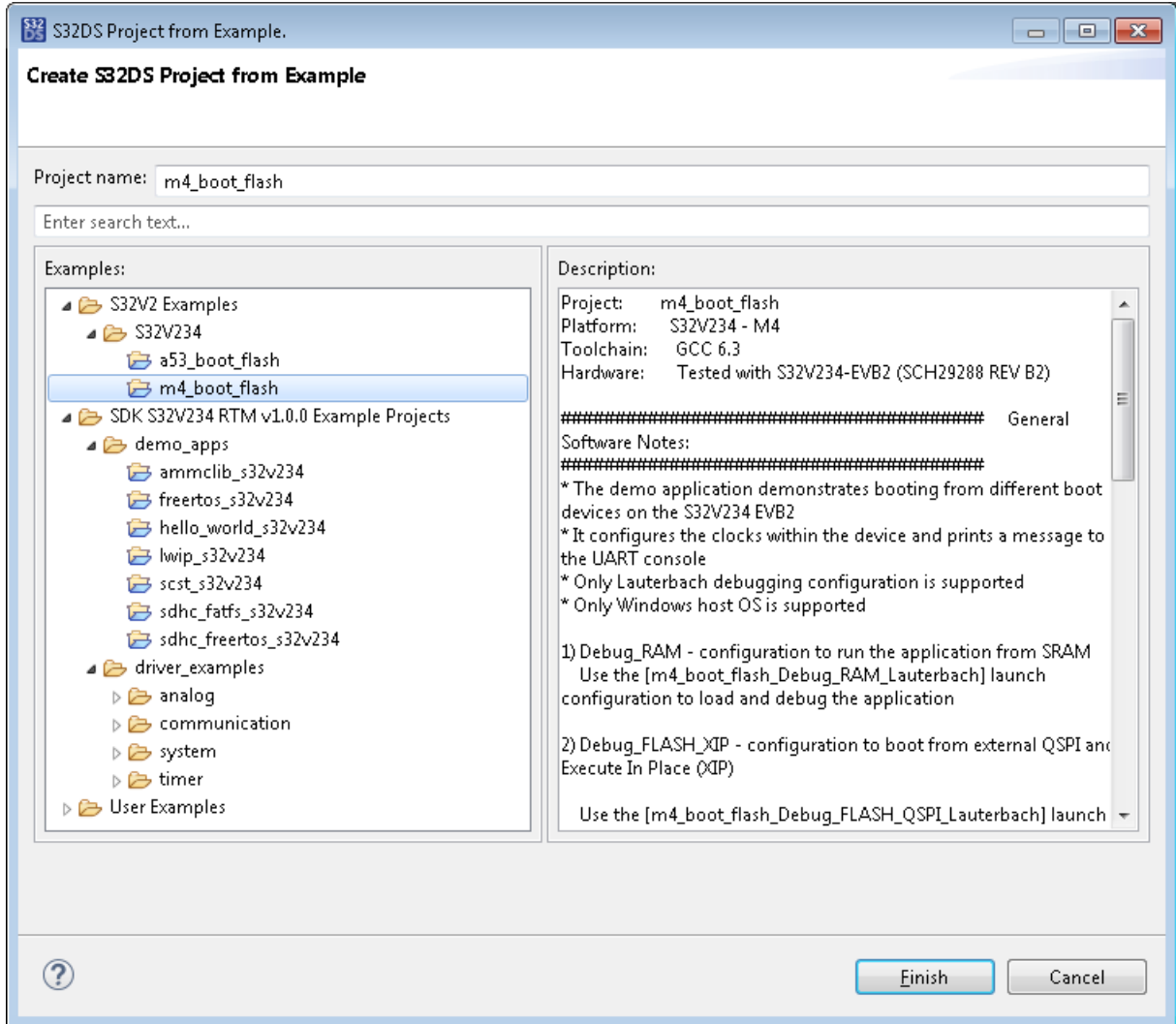


If the target processor has multiple cores, the wizard generates a dedicated project for each core that you have selected on the second page.

Creating a project from an example

You can create a new project on the basis of a sample project. The new project includes all files of the sample project. To create a new project from an example project:

1. Launch the **S32DS Project from Example** wizard by clicking **File > New > S32DS Project from Example** on the menu.



2. The wizard displays all sample projects in the **Examples** pane. Click a project that fits best. To find a certain project, start typing the project name in the search box above the **Examples** pane.
3. In the **Project name** field, specify a unique name of the new project. Use alphanumeric characters (A-Z, a-z, 0-9) and underscores ('_'). Do not start the project name with a digit.
4. Click **Finish**. The new project appears in the **Project Explorer**.

If system generates any errors or warnings, you can see them in the **Problems** view. You can use the **Quick Fix** option if possible:

- Right-click an error message and select **Quick Fix** option from the context menu,
- Select errors to fix,
- Click **Finish**.

Errors will disappear and project will be available for build and debug.

Importing a project

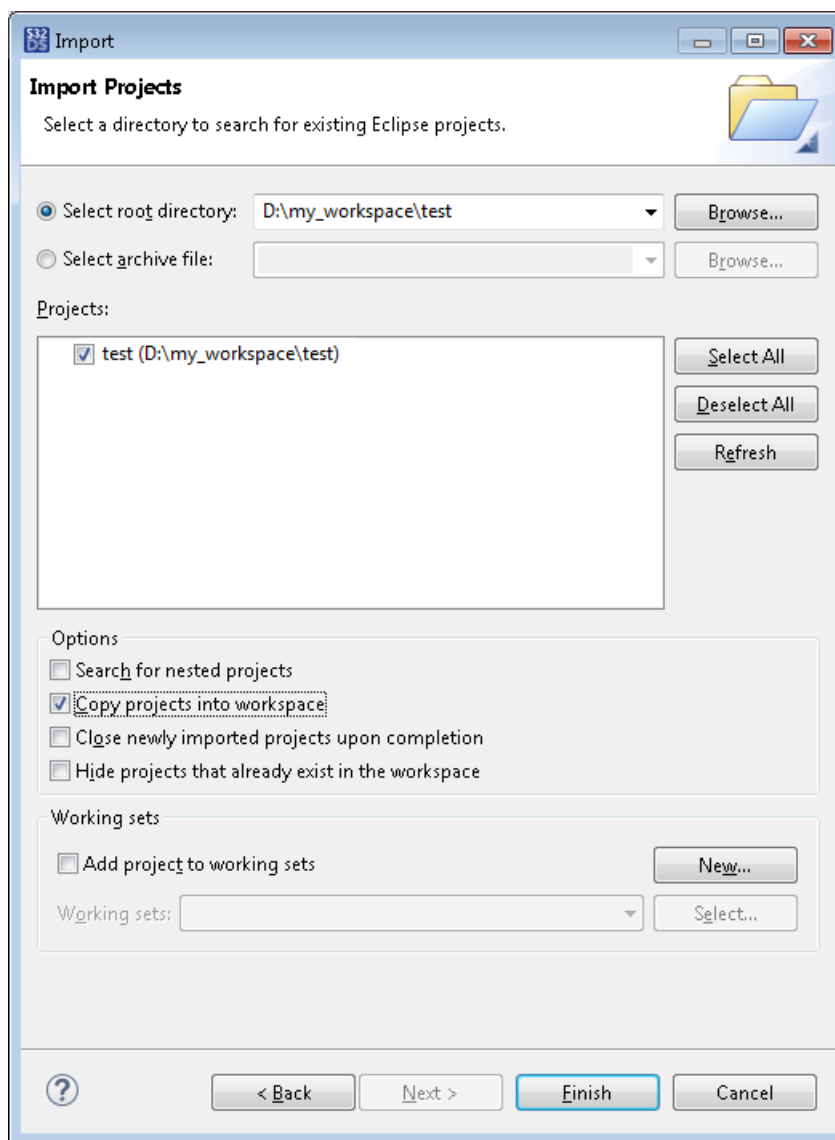
You can add an existing project to your workspace by importing it from:

- a system folder,
- an archive file,
- a ProjectInfo.xml file.

Importing a project from a system folder or an archive file

To import a project into the workspace:

1. Click **File > Import... > General > Existing Projects into Workspace** on the main menu, then click **Next**.
2. In the **Import Projects** dialog box, browse to the projects' root folder or archive file.

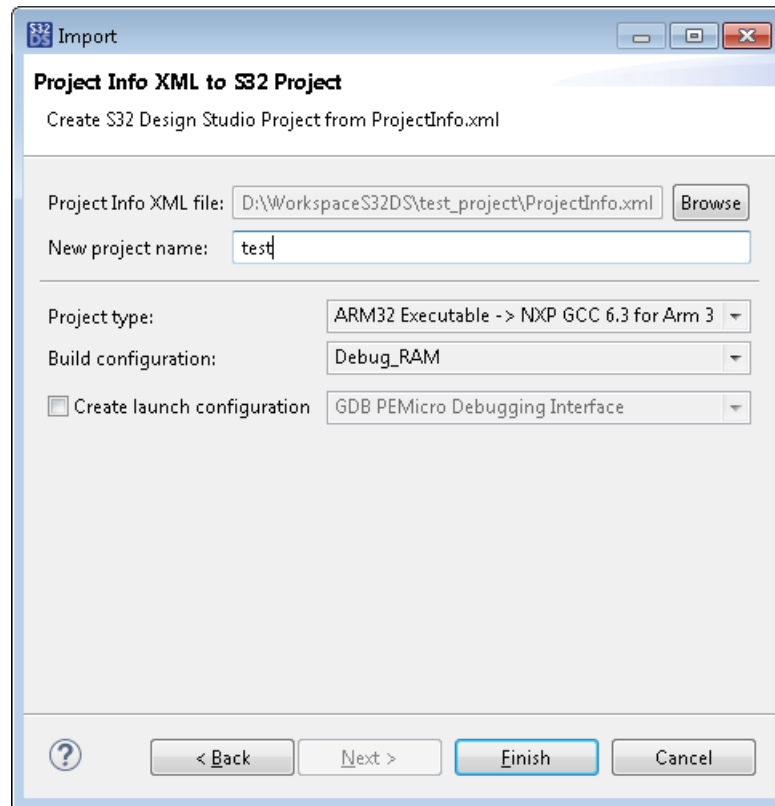


3. In the list of folders, select the projects to be imported.
4. Select the **Copy projects into workspace** option to not affect the original project.
5. Click **Finish**.

Importing a project from a ProjectInfo.xml file

To import a project into the workspace:

1. Click **File > Import...** on the main menu.
2. In the **Import** dialog click **S32 Design Studio for S32 Platform > ProjectInfo XML as S32DS Project**, then click **Next**
3. Browse to the ProjectInfo.xml file.



4. Specify the project name.
5. In the **Project type** list select the toolchain available for the core.

Note: Be careful to select the appropriate core type toolchain, f.e.: if importing project with linux core type, select the linux configuration. Selecting the wrong toolchain during import may lead to unavailability to compile the project.

6. In the **Build configuration** list select the configuration available for the project type and core.
7. If you need a new launch configuration for debugging enable the **Create launch configuration** option, select the debug configuration type from the list.
8. Click **Finish**.

If imported with the **Create launch configuration** option enabled, the **Debug Configurations** dialog will appear with a new configuration created and the **Debugger** tab displayed. For debugger settings details refer to topic [Debugger tab](#).

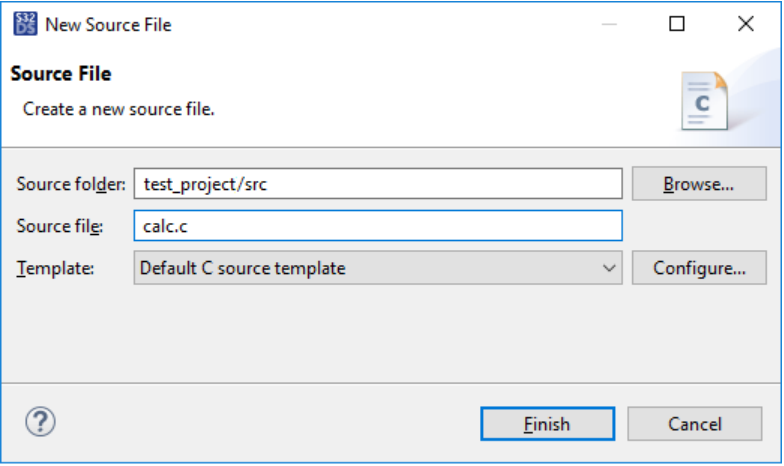
Adding files and folders to a project

To add files and folders to a project in the **Project Explorer**, use any of these options:

Table 2: Adding files and folders to a project

To do this...	... take the following steps:
Drag and drop a file or a folder from the file system	<ol style="list-style-type: none"> 1. Select a file or a folder in the system window and drag it to the Project Explorer. 2. Drop the file or the folder to the destination project folder.
Copy and paste a file or a folder from the file system or from the Project Explorer	<ol style="list-style-type: none"> 1. Select a file or a folder in the system window or in the Project Explorer and copy it to the clipboard using CTRL+C. 2. In the Project Explorer, select the destination project folder and paste the copied file or folder using CTRL+V. The copied objects are placed inside the selected project folder.
Import files from the file system	<ol style="list-style-type: none"> 1. Click File > Import on the main menu. 2. In the Import wizard, go to General > File System. Click Next. 3. Specify the settings for import: <div data-bbox="688 743 1370 1549" data-label="Image"> </div> <ul style="list-style-type: none"> • In the From directory field, browse to the folder where the files for import are located. • Flag files for import in the right pane, or flag the parent folder in the left pane to import all included files. • Specify the target folder for import in the Into folder field. • To import files with their parent folder, select the Create top-level folder option. 4. Click Finish.

To do this...	... take the following steps:
<p>Add a new folder to a project</p>	<ol style="list-style-type: none"> In the Project Explorer, click the project folder where you need to nest a new folder. Click File > New > Folder on the main menu. In the New Folder dialog box, enter the name of the new folder. <div data-bbox="712 386 1344 1163" data-label="Image"> </div> <ol style="list-style-type: none"> Click Finish.
<p>Add a new file to a project</p>	<ol style="list-style-type: none"> In the Project Explorer, click the project folder where you need to add a new file. On the main menu, click File > New and the option required - Source File, Header File, or File from Template (a TXT file). In the New File dialog box, point the source (parent) folder where the new file will be added, enter the file name, and select the file template from the list.

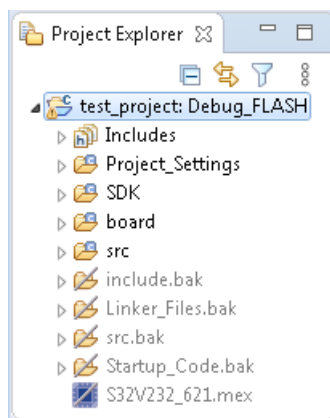
To do this...	... take the following steps:
	 <p>4. Click Finish.</p>

Adding a device configuration

The integrated S32 Configuration Tool enables you to create and develop a device configuration within a project.

A device configuration sets up interconnections between the device's pins and clocks and connected peripherals. A typical configuration specifies pin electrical features, signal routing from pins to peripherals, muxing, clock element settings, and clock output frequencies.

To store a device configuration, the S32 Configuration Tool generates a MEX configuration file and the empty board folder for the source files.



The topics in this section describe how to work with a device configuration in a project:

- [Creating a project with a device configuration](#)
- [Editing a device configuration](#)
- [Importing a device configuration](#)
- [Exporting a device configuration](#)

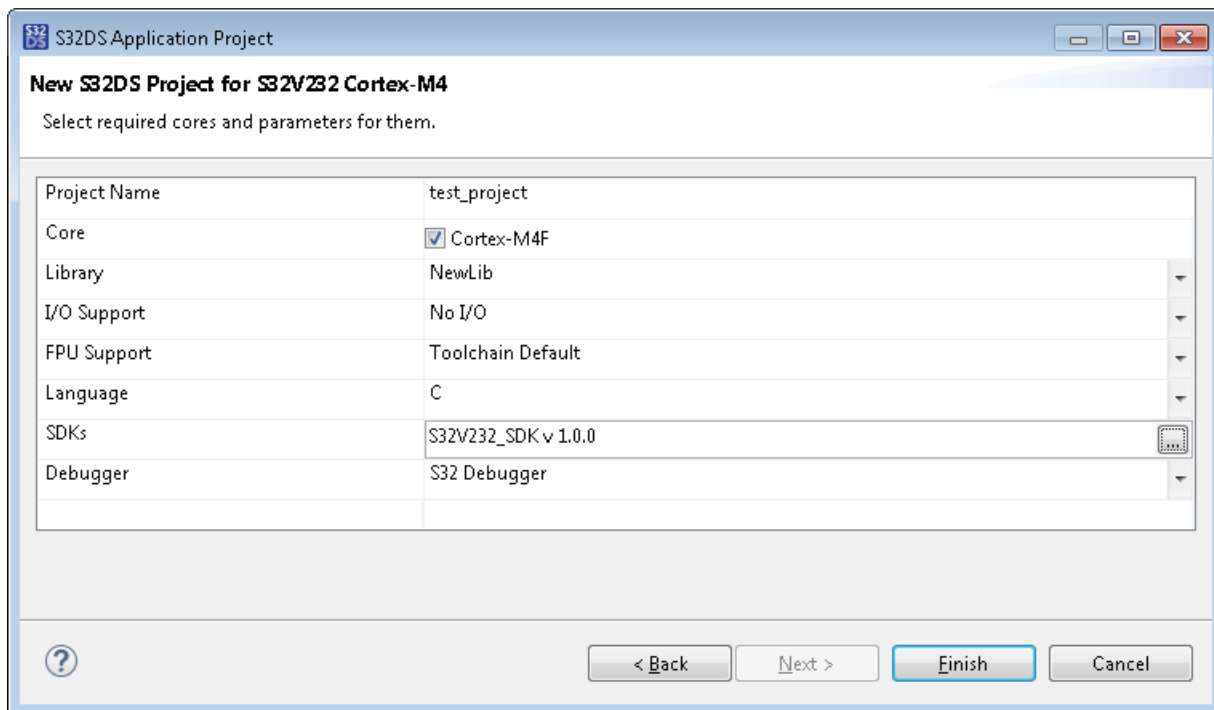
To learn more about the S32 Configuration Tool, refer to the *S32 Configuration Tool Getting Started* guide. This document can be reached from the **Help > Help Contents** menu.

Creating a project with a device configuration

To create a project with a device configuration:

1. On the main menu, click **File > New > S32DS Application Project**.
2. In the project creation wizard, select the processor family and the core and specify a unique project name. Use alphanumerics (A-Z, a-z, 0-9) and underscores ('_'). Do not start the project name with a digit.
3. Click **Next**.
4. On the next page, define the required cores and parameters. Select an SDK to be used.

Note: The **S32 Configuration Tool** is enabled by default when an SDK is selected for the project. If SDK is not selected the **S32 Configuration Tool** is not accessible for the project.




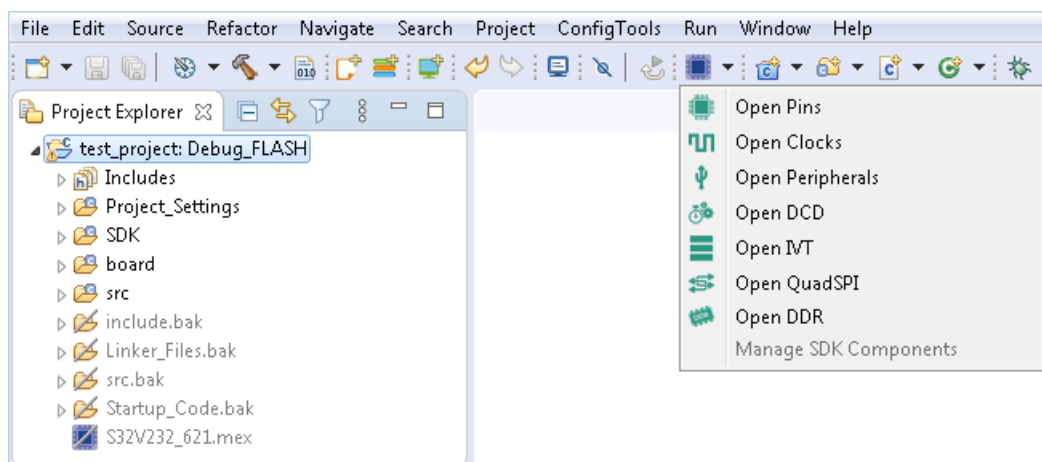
5. Click **Finish**. The new project appears in the **Project Explorer**.

Editing a device configuration

To let you edit a device configuration, S32 Design Studio provides perspectives named **Pins**, **Clocks**, **Peripherals**, **DCD** (Device Configuration Data), **IVT** (Image Vector Table), **QuadSPI** and **DDR**. The settings that you edit and save in these perspectives are automatically stored in the configuration file of the currently used project.


To edit a device configuration in a perspective, do any of the following:

- Double-click the MEX file in the **Project Explorer**. By default, you will get to the **Pins** tool.
- Select the project in the **Project Explorer** and click the  (*Open S32 Configuration*) button on the toolbar. Click to open the required perspective.



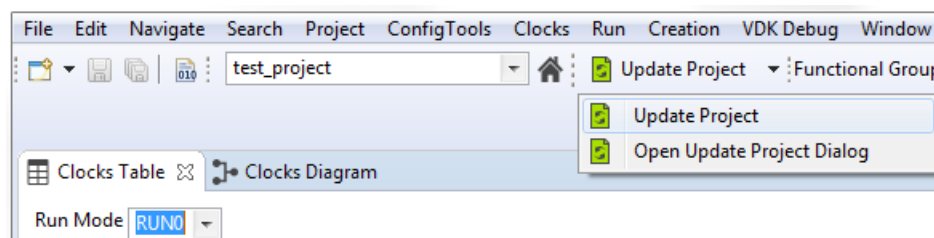
- Right-click the project in the **Project Explorer** pane. On the drop-down context menu, click **S32 Configuration Tool** and the required option.

The use of settings in the perspectives is beyond the scope of this document. You can read more about the subject in the *S32 Configuration Tool Getting Started* guide. To open this guide, click **Help > Help Contents** on the menu.

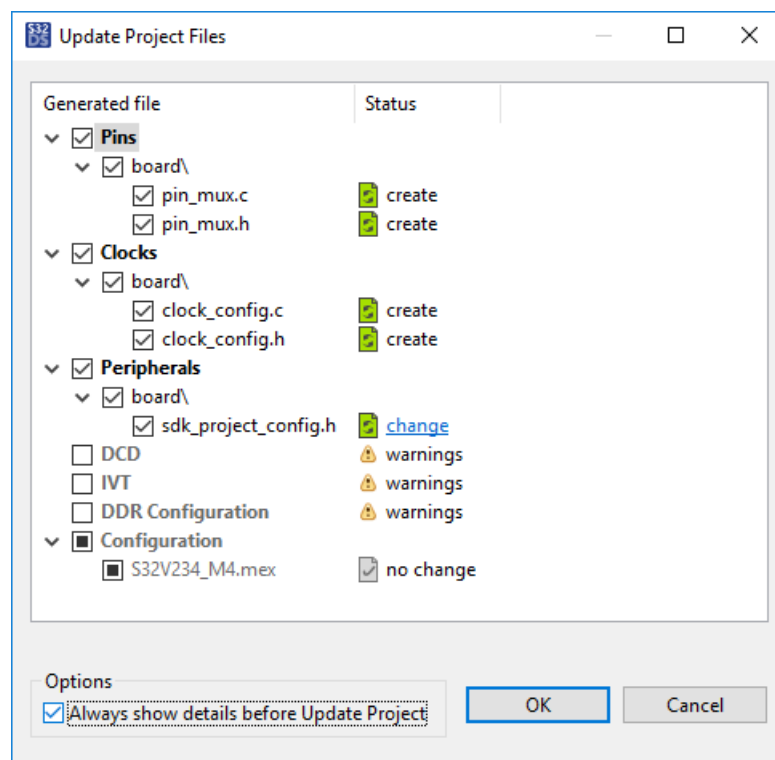
While configuring your processor, you can switch between the perspectives using the  (*Open S32 Configuration*) toolbar button.

To save the changes to the project, use the **Update Project** toolbar button. A click on this button either saves the latest device configuration silently, or opens the **Update Project Files** dialog box.

When saving the changes for the first time, click **Update Project > Open Update Project Dialog**.



In the **Update Project Files** dialog box, select the device configuration files to be created in the project and updated with the latest pin, clock, and peripheral settings:



When you click **OK**, the project will be updated as follows:

- If a file is labeled “create”, it will be created in the project's board folder.
- If a file is labeled “no change”, there are no updates to be saved to this file.
- If a file is labeled “change”, the existing file will be updated in the board folder. Click the “change” label to compare the file before and after update.
- If a file is not selected, it will be skipped from update, or not created.

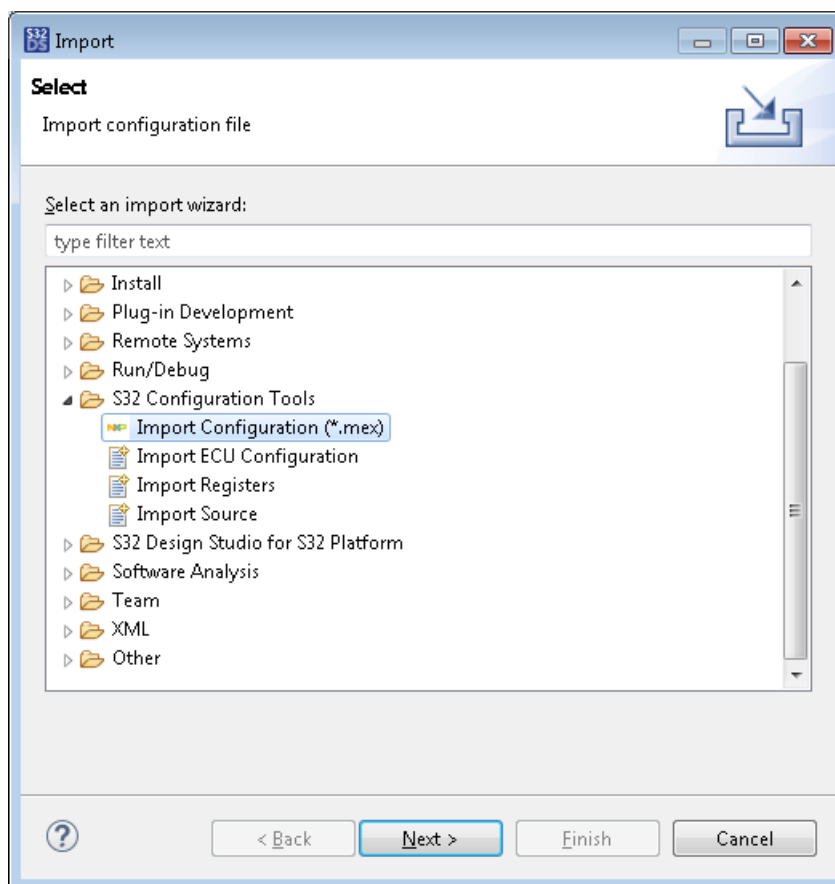
You can configure the **Update Project** button to always open the **Update Project Files** dialog box. To do it, select the **Always show details before Update Project** option in the **Update Project Files** dialog box, then click **OK**.

Importing a device configuration

A device configuration includes a MEX file and source code files that specify interconnections between the device's pins and clocks and connected peripherals. You can create these files within your project, or you can import external configuration files into your project.

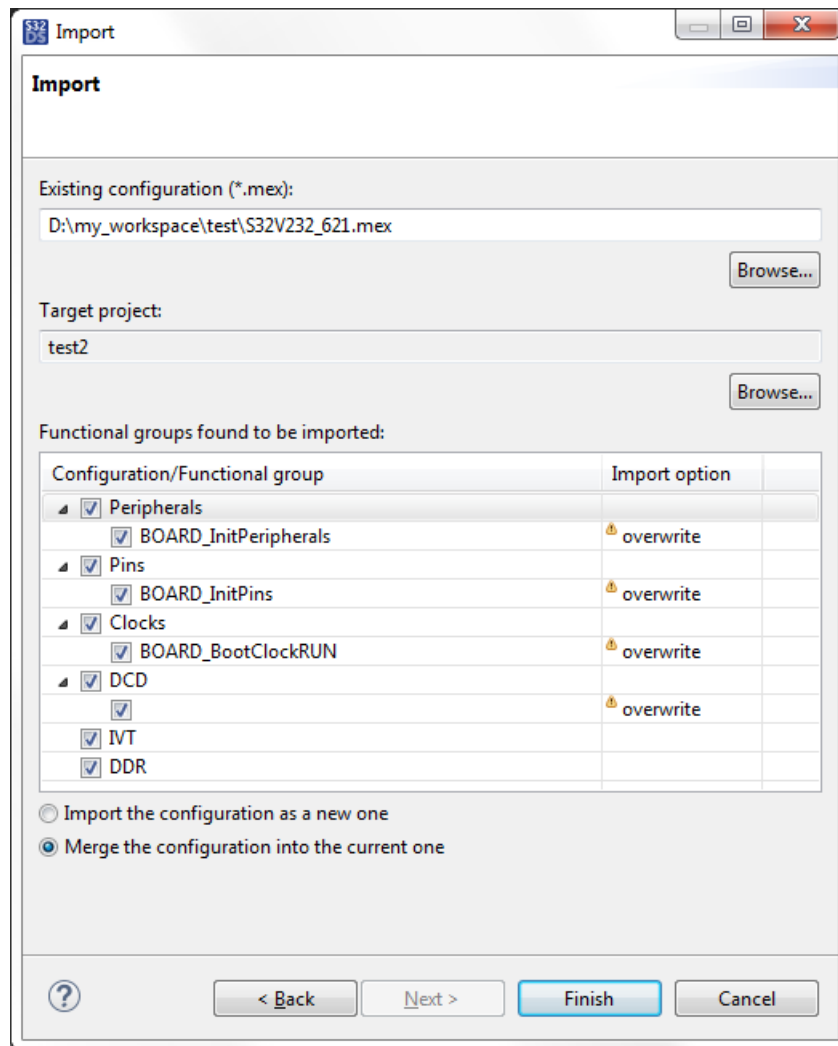
To import device configuration files:

1. Create or open a project that includes a device configuration (MEX file).
2. Select the project in the **Project Explorer** and click **File > Import** on the main menu.
3. In the **Import** dialog box, double-click **S32 Configuration Tool**. Click to import a MEX file, registers, or source files, then click **Next**.

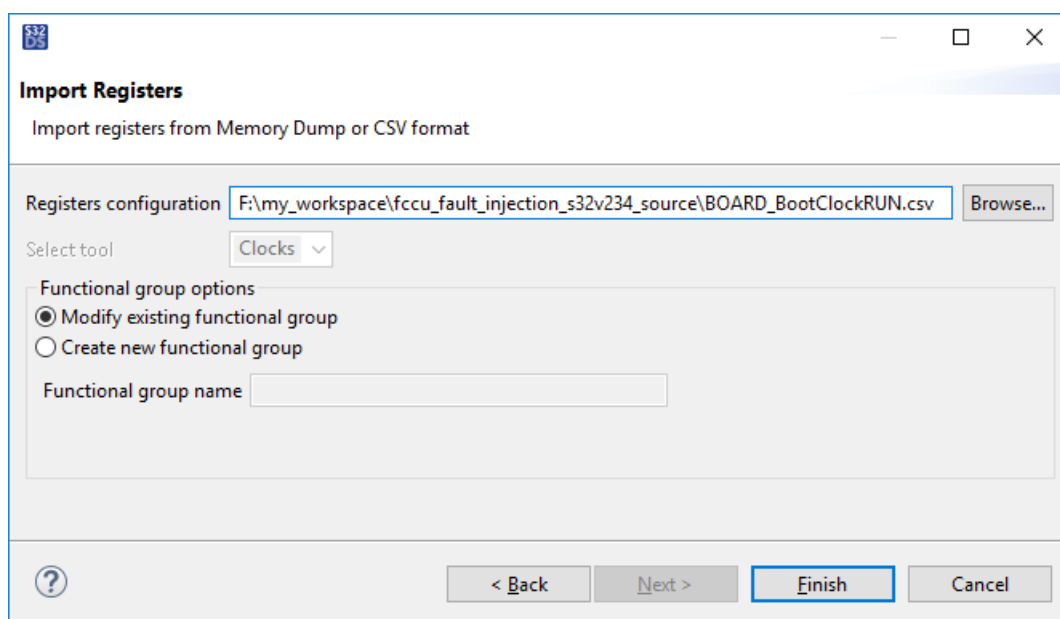


4. Specify the settings for import:

- If you import a MEX configuration, browse to the existing MEX file, specify the target project for import, and select tool-specific data to be imported:

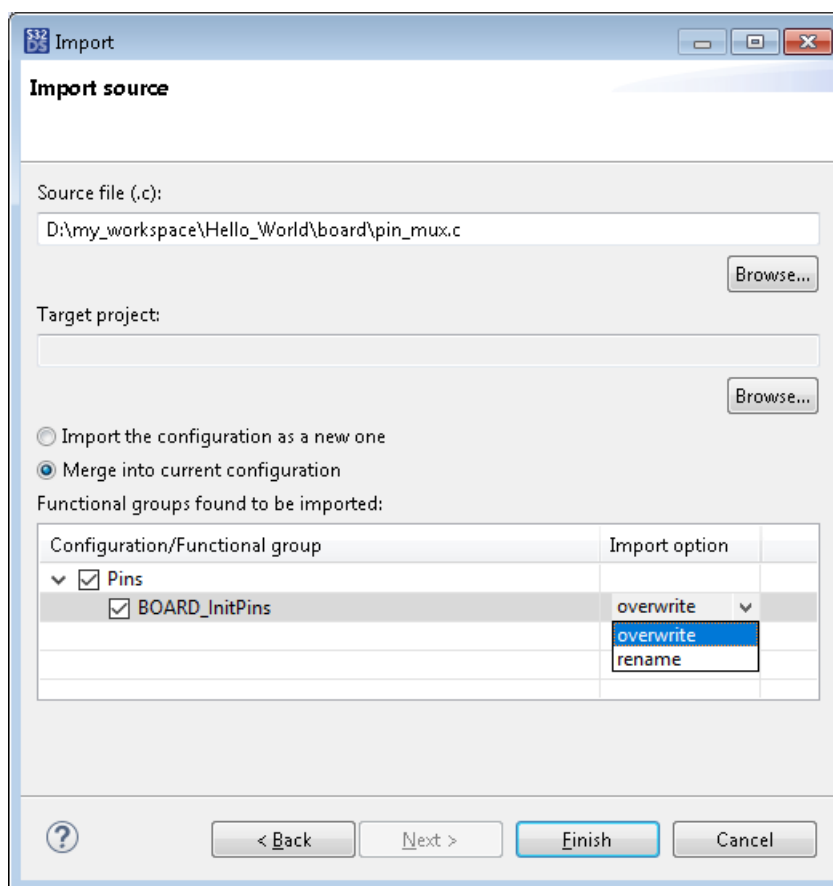


- If you import registers, browse to the registers configuration (for instance, to a CSV file), and specify the functional group under which the registers will be added:



Note: To learn how to get a CSV file with registers configuration, refer to [Exporting a device configuration](#).

- If you import a source file, browse to it. Then select functional groups to be imported.



If the source file already exists in the target project, specify how to handle the existing functions when their names match the names of functions to be imported:

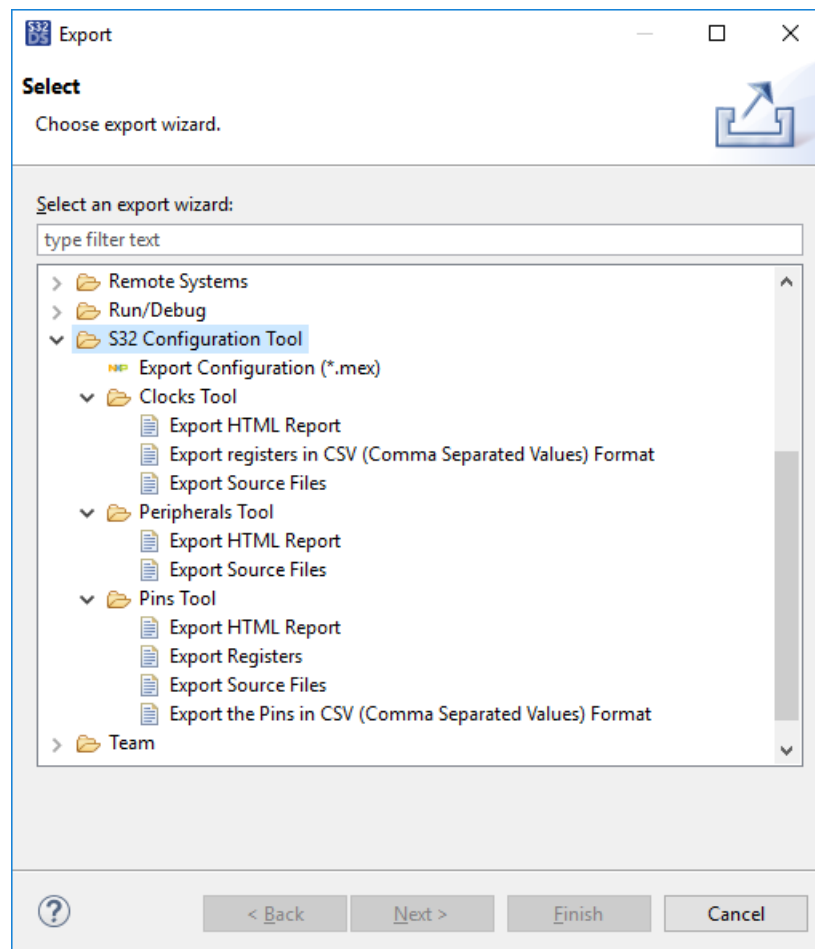
- Select **overwrite** to replace the existing function with the imported one.
- Select **rename** to keep in the destination source file both the existing function and the imported function under different names.

5. Click **Finish**.

Exporting a device configuration

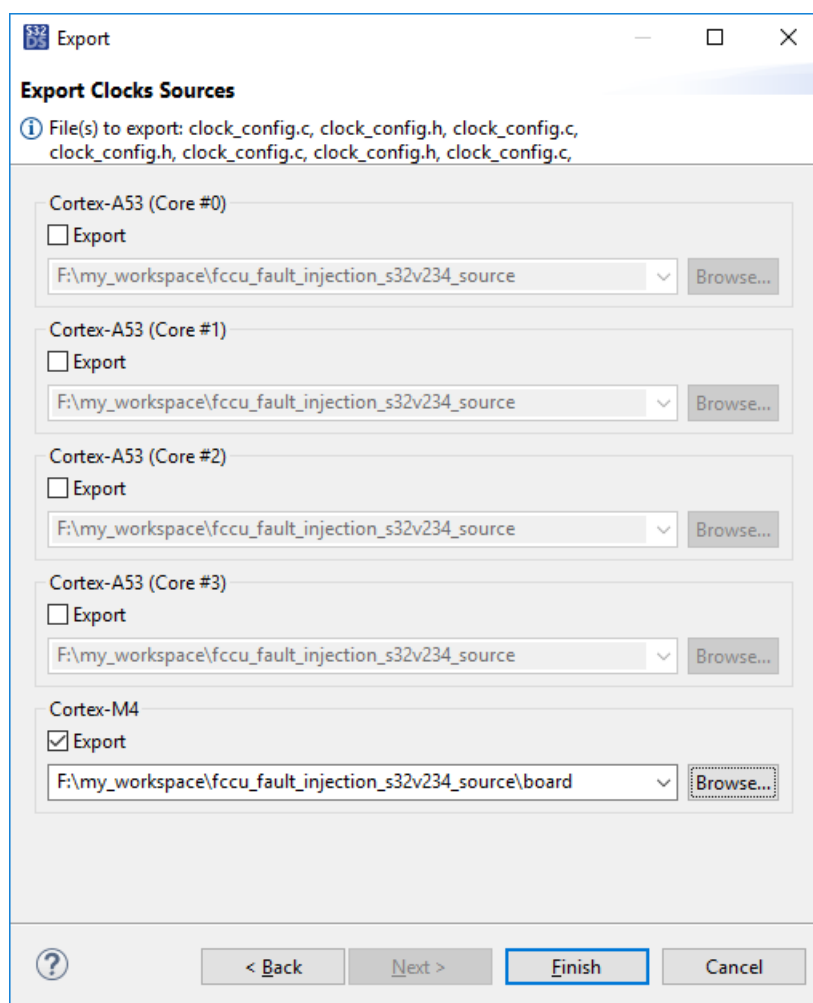
To export device configuration files:

1. Open a project with the implemented device configuration.
2. Select the project in the **Project Explorer** and click **File > Export** on the main menu.
3. In the **Export** dialog box, double-click **S32 Configuration Tool**. Click to export the MEX file, or expand the tool folder and click to export an HTML report, registers, pins, or source files. Click **Next**.



4. If you export an HTML report, registers, or pins, browse to the destination folder. For the MEX file, specify the file name and the destination path.

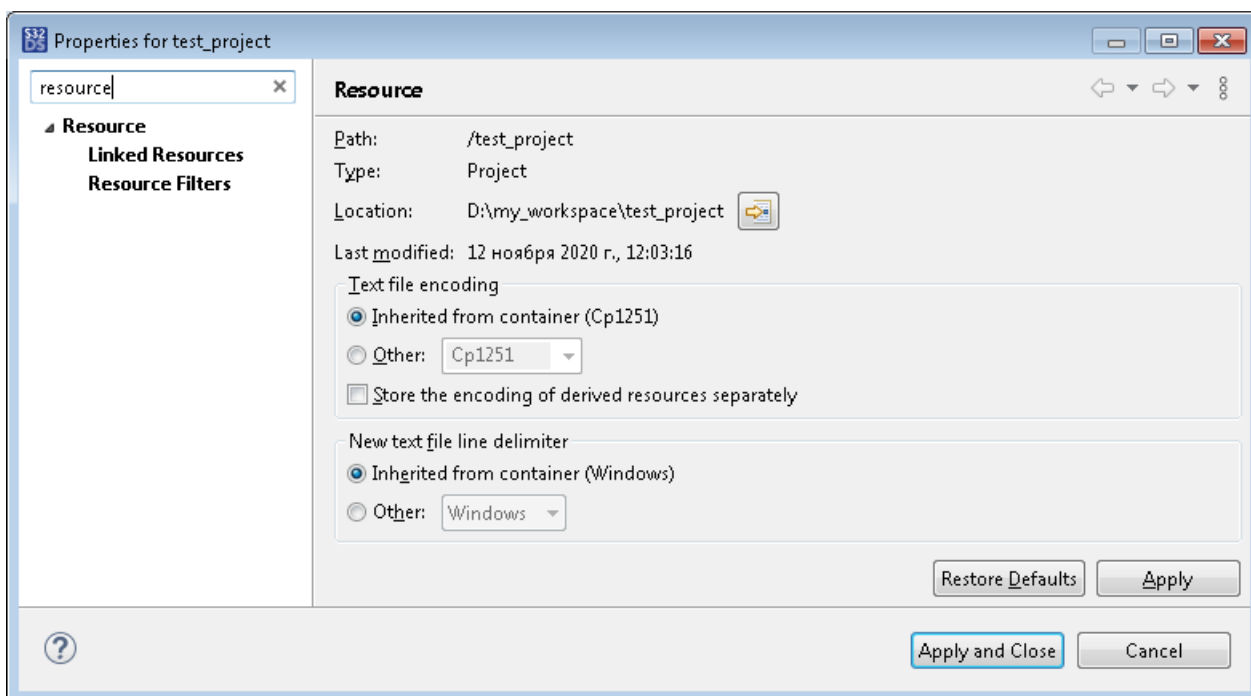
If you export the source files, click the cores which files you need to export. For each core, specify the destination folder (for instance, the board folder in a different project):




5. Click **Finish**.

Locating project files and folders in the file system

To locate a project file or folder in the file system, right-click it in the **Project Explorer** and click **Properties**. Click **Resource** in the **Properties** dialog box. The absolute path of the selected file or folder is provided in the **Location** field:



Click the  (*Show In System Explorer*) button to open the selected folder in the system dialog box. If you selected a file, this button opens the parent folder containing this file.

Renaming a project

To rename a project, open it in the **Project Explorer**. Before you proceed with renaming, make sure to resolve all the project-related issues reported in the **Problems** view. Also, clean the project from any artifacts belonging to previous builds:

- Right-click the project in the **Project Explorer** and click **Clean Project** on the context menu.
- Delete the **Debug** folder and other folders with the build output (if any available).

To rename a project:

1. Select the project in the **Project Explorer** and click **File > Rename** on the menu or press **F2**.
2. Specify the new project name in the **Rename Project** dialog box. The project name must be unique across the workspace. The allowed characters are alphanumeric (A-Z, a-z, 0-9) and underscores ('_'). Project names starting with a digit are not allowed.
3. Click **OK**.

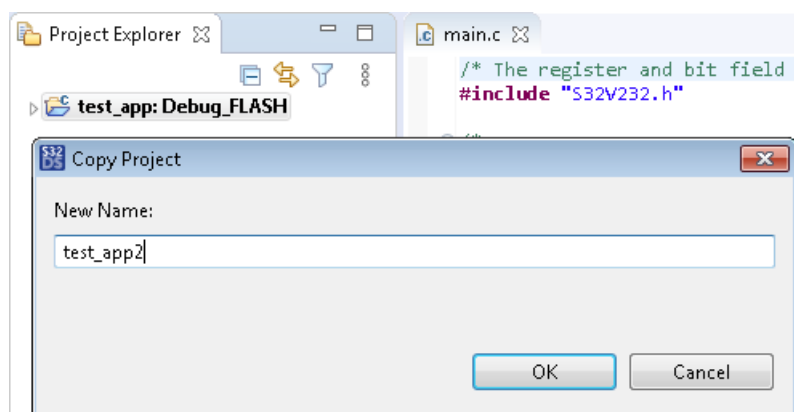
The project is displayed in the **Project Explorer** with the new name and with the project files, functions, and other resources renamed accordingly.

Duplicating a project

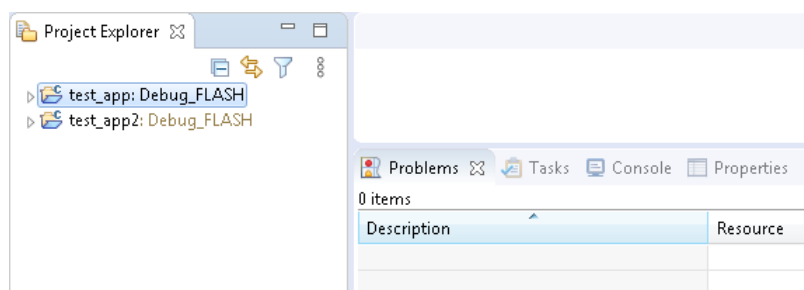
If necessary, you can save a copy of an existing project in the workspace under a different name. The duplicated project includes all files, folders and resources of the original project renamed accordingly.

To duplicate a project in the workspace:

1. Select the original project in the **Project Explorer** and press **CTRL+C**.
2. Press **CTRL+V** to paste the copied project to the workspace. Rename the duplicated project and click **OK**.



The new project appears in the **Project Explorer**.



Saving a project to User Examples

S32 Design Studio for S32 Platform is installed with a collection of project examples that demonstrate programming concepts and use of SDKs. A project example can be copied to the workspace and reused as described in topic [Creating a project from an example](#).

All project examples are available for use in the [S32DS Project from Example wizard](#). To open the wizard, click **File > New > S32DS Project from Example**.

The **User Examples** section of the **S32DS Project from Example** wizard is reserved for project examples added by the user. To add a project to **User Examples**, copy the project folder from your workspace to the {S32 Design Studio for S32 Platform 3.4 installation path}/S32DS/examples folder.

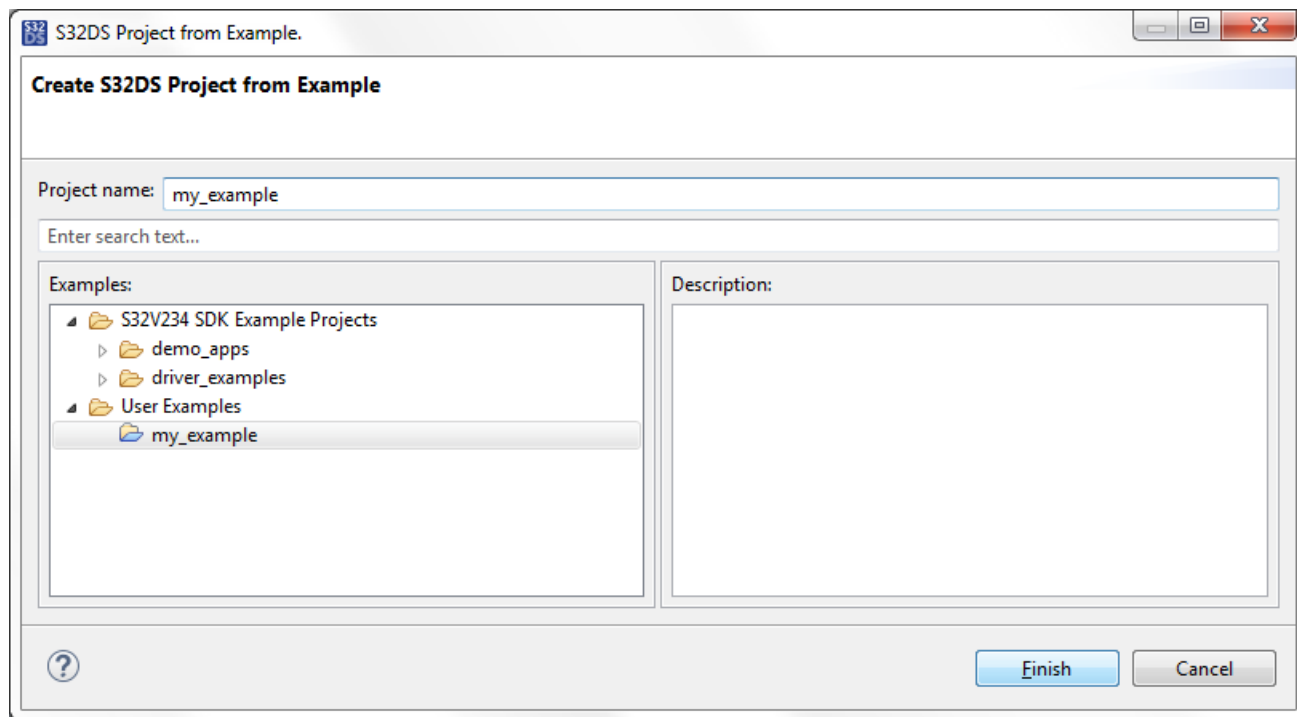
An example can include several projects. For instance, when you design an application for a multi-core processor, you use a separate application project for each core. You can pack all these projects into one example. When someone decides to use your example, they install all included projects to the workspace.

To add a multi-project example to **User Examples**:

1. In the **Project Explorer**, click an application or library project that you want to add to user examples. Click **Project > Properties** on the main menu.
2. In the **Properties** dialog box, click **Project References** in the left pane. Make sure that all other projects that you want to include in your example are referenced.
3. In the **Properties** dialog box, click **Resource** in the left pane. In the right pane, click the **Show in System Explorer** button to open the project folder.
4. In the workspace, all projects of a multi-core solution are located in the upper-level folder named similarly. The folder was added by the project creation wizard. Copy and paste this folder to the following location: {S32DS 3.4 installation folder}/S32DS/examples/.

Note: If the common folder for your projects is missing in the workspace, create a folder with a proper name in /S32DS/examples/ and copy all projects of your example to this folder.

Now the **S32DS Project from Example** wizard displays your example under **User Examples**.



The example is located in the upper-level folder that you copied or created in the /S32DS/examples folder.

Exporting a project

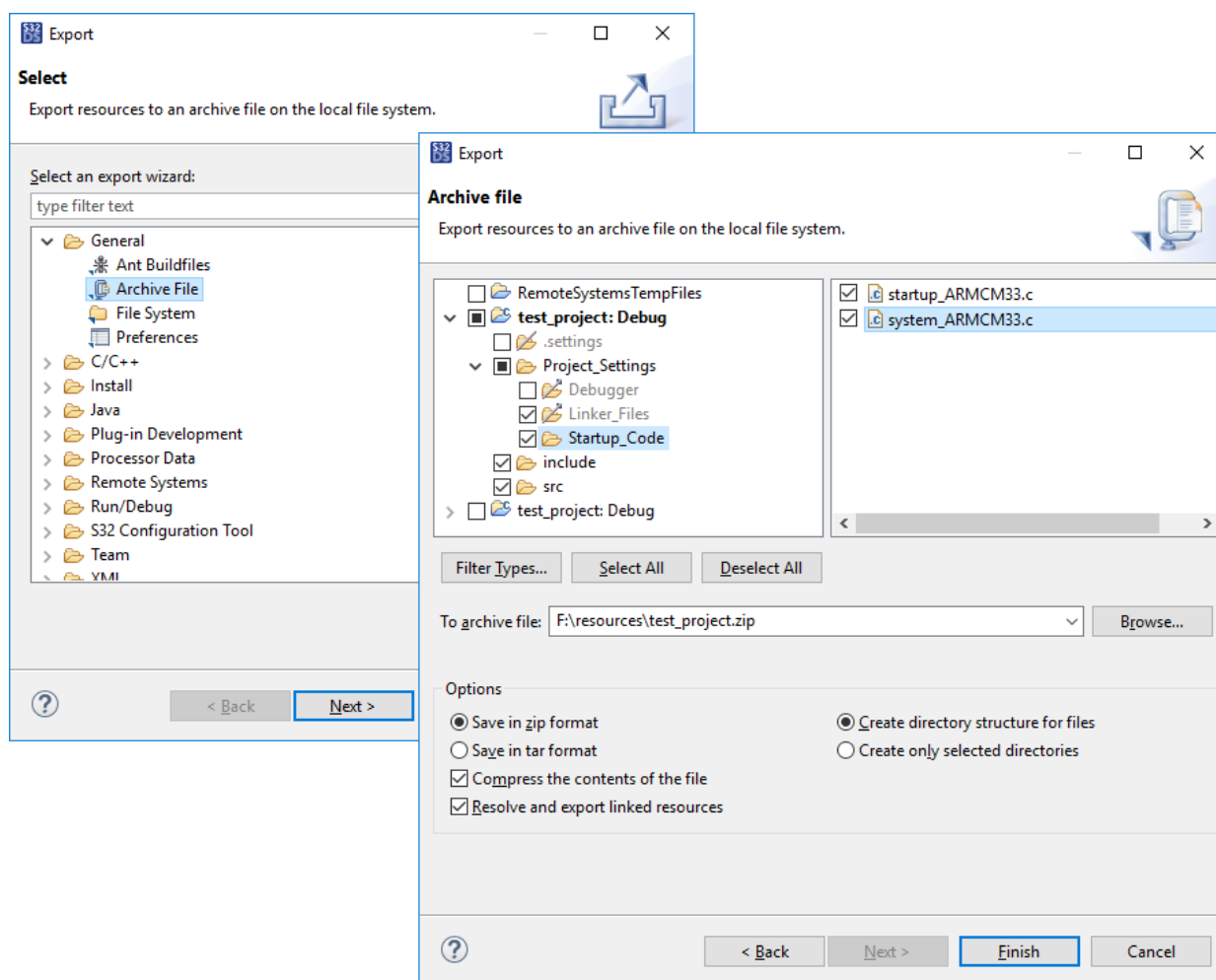
You can export an S32DS project to:

- a system folder,
- an archive file - ZIP or TAR,
- a ProjectInfo.xml file.

Exporting a project to a system folder or an archive file

To export a project to a system folder or to a ZIP or TAR archive file:

1. Right-click the project in the **Project Explorer** and click **Export** on the context menu.
2. In the **Export** wizard, go to **General** and select **Archive File** or **File System** as the destination for export. Click **Next**.



3. On the next wizard page, do the following:

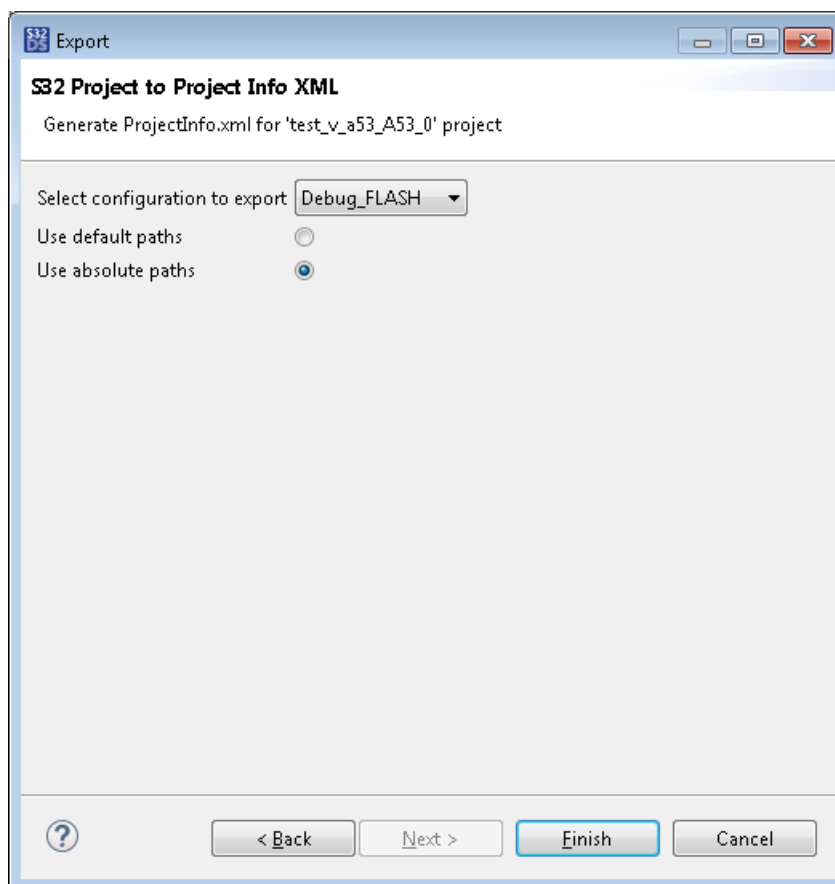
- In the left pane, remove flags from the project folders that you want to be excluded from export. Click a folder in the left pane to see all included files in the right pane. Remove flags from the project files to be excluded from export.
- Browse to the destination folder or archive file, or enter the path manually. If the specified folder or archive file does not exist, it will be created during export.
- Specify the options for export.

4. Click **Finish**.

Exporting a project to a ProjectInfo.xml file

To export a project to a system folder or to a ZIP or TAR archive file:

1. Right-click the project in the **Project Explorer** and click **Export** on the context menu.
2. In the **Export** wizard, click **S32 Design Studio for S32 Platform > S32DS Project as ProjectInfo XML** and click **Next**.



3. Select the configuration to export.
4. Choose to use default or absolute paths.
5. Click **Finish**.

The data is exported to the ProjectInfo.xml file in the project tree and its contents are shown in the *editor area*.


Closing and reopening a project

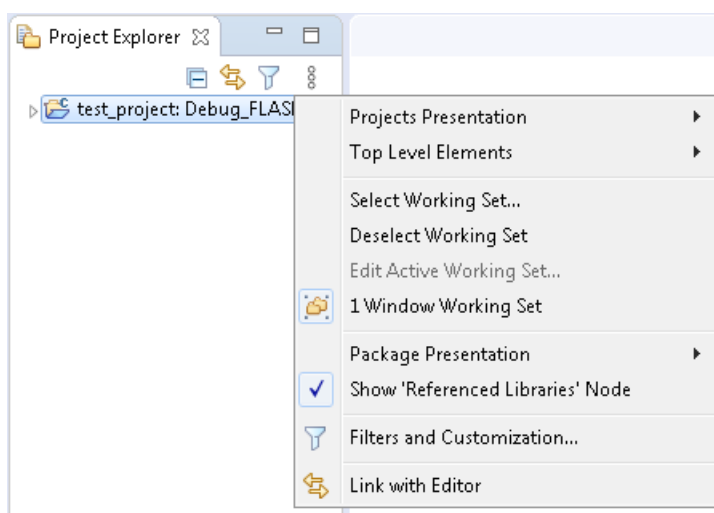
If no activity is happening in a project for a while, you may prefer to close it. A closed project still displays the root folder in the **Project Explorer**, but you cannot expand it to see the project files and folders. A closed project cannot be modified, corrupted, or involved in group operations such as building all projects.

To close a project, select it in the **Project Explorer** and click **Project > Close Project** on the menu. To close all projects except the selected one and its related projects, right-click the selected project and click **Close Unrelated Projects** on the context menu.

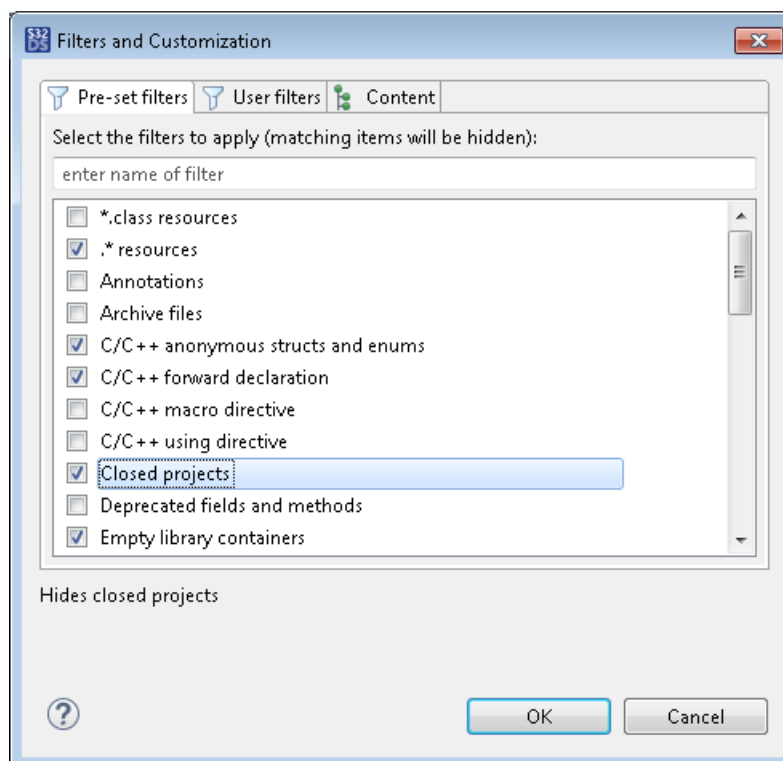
To reopen a closed project, select it in the **Project Explorer** and click **Project > Open Project** on the menu.

If necessary, the **Project Explorer** can be customized to hide closed projects:

1. Click the **View Menu** toolbar button and click **Filters and Customization** on the context menu or simply push  toolbar button.



2. In the **Filters and Customization** dialog box, go to the **Pre-set filters** tab and flag the **Closed projects** option. Click **OK**.



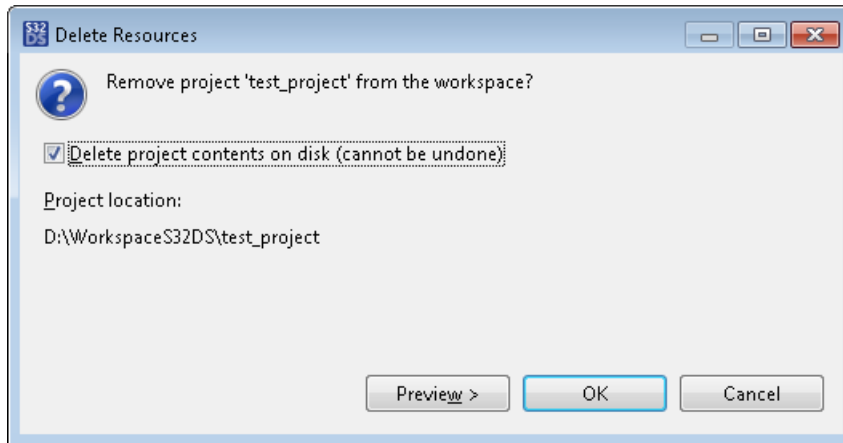
To reopen a closed project hidden from the **Project Explorer**, customize the **Project Explorer** settings to show closed projects. Then reopen the closed project as described above.

Removing a project

When a project is finished and saved to a storage, you may prefer to remove it from the workspace and permanently delete it from the disk.

To remove a project:

1. Right-click the project in the **Project Explorer** and click **Delete** on the context menu.
2. In the **Delete Resources** dialog box, select the **Delete project contents...** option if you prefer to permanently delete the project files from the disk. Click **OK**.



If you have removed the project with the **Delete project contents...** option not selected, the project can be restored in the workspace as described in topic [Importing a project](#). Otherwise, the removal cannot be reverted.

Building projects

Overview

Building code in S32 Design Studio for S32 Platform is highly configurable and provides many flexibilities. In the simplest case, you just click the menu button and get the compiled file for debugging in a couple of moments. This scenario uses the default build settings specific to your project.

To have a deeper view and a better command of the build process, learn about the basic concepts described below.

Build targets

The build target is an output file compiled from the project code at build time. Depending on the project type, the build generates an executable file or a library file ready for execution. In addition, you can generate secondary output such as an image file, a disassembly file, or other.

Build configuration

The settings that configure the build process are collected in a *build configuration* - a special section of the project properties. A build configuration specifies the builder to be used, the prioritized list of build tools available to the builder, and the tool settings. Also, a build configuration determines the project folders and files participating in the build, and enables build logging.

A project in S32 Design Studio for S32 Platform can have multiple build configurations. When just created, a project gets several default configurations for building the debug and release versions of the code. You can create more configurations for a project, adjust the settings in any build configuration, rename the build configurations, and delete any configuration from the project.

One of the project's build configurations is selected to be active. The active build configuration is the one to be used at build time.

Builder

The builder is a component that calls the build tools to generate the intermediate build files and the build target. In the build configuration, you select the internal CDT builder or an external builder supported in S32 Design Studio for S32 Platform. The internal builder does not create a makefile, while the external builders do.

This chapter focuses on building by means of an external builder.

Toolchains

S32 Design Studio for S32 Platform comes with a collection of toolchains targeted at different project types. Most of the supported toolchains are based on the standard GCC tools and implement extensions to comply with a target hardware architecture.

When you create a project, you specify the toolchain that matches the project's target MCU and the core. After that, you should not select a different toolchain for the project.

The tools included in the toolchain are set up by default and can be fine-tuned in the build configuration.

Build tools

The build tools included in a toolchain will be called by the builder to produce the build output:

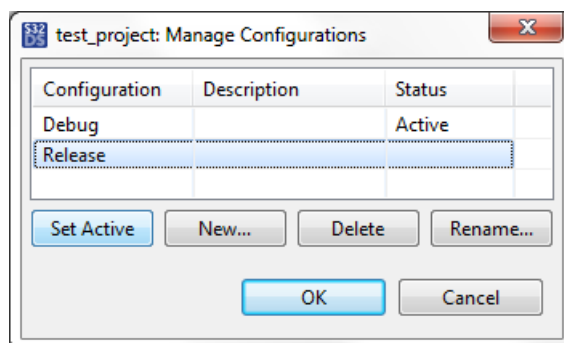
- The basic tools are called when the build is started. These tools are used to generate the build target. The call order (the compiler, the assembler, and the linker) is predefined and cannot be altered; no tool in this sequence can be skipped.
- The secondary (or *optional*) tools are executed after the build target has been generated. These tools are used to create secondary build output such as the flash image of the application, the disassembly listing, and so on. To be called at build time, each secondary tool needs to be enabled and configured in the active build configuration.

Besides, a toolchain can include the disassembler tool and the preprocessor tool that are never called at build time. You can call these tools from the menu, without starting the build, to look into the assembler instructions, string substitutions and other conversions that will take place later when you build the project.

Using build configurations

Any project in S32 Design Studio has at least one build configuration. To view all build configurations belonging to a given project, open the **Manage Configurations** dialog box in any of these ways:

- Click the project name in the **Project Explorer**, then click **Project > Build Configurations > Manage** from the main menu.
- Right-click the project name and click **Build Configurations > Manage** from the context menu.
- Right-click the project name and click **Properties** from the context menu. In the project properties, click **C/C++ > Build** in the left pane, then click the **Manage Configurations** button in the right pane.



To perform an action from the **Manage Configurations** dialog box, select a build configuration from the list and click the respective button:

- **Set Active:** Make the selected build configuration active, that is, used in project builds by default. Find the details in [Setting the active build configuration](#).
- **New:** Create a new configuration for the project. Find the details in [Creating a build configuration](#).

- **Delete:** Delete the selected configuration from the project. If you delete the active configuration, the “active” status is automatically assigned to the top configuration in the list. When there is one build configuration in the list, the **Delete** button is disabled.
- **Rename:** Rename the selected build configuration.

To learn how to view and edit the settings in a particular build configuration, refer to [Editing a build configuration](#).

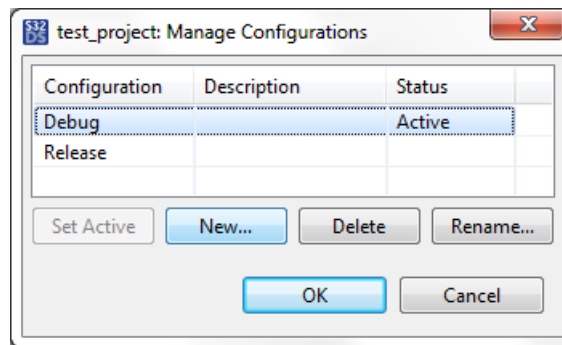
To learn how to view and manage the project files and folders in all build configurations, refer to [Managing project resources in build configurations](#).

Creating a build configuration

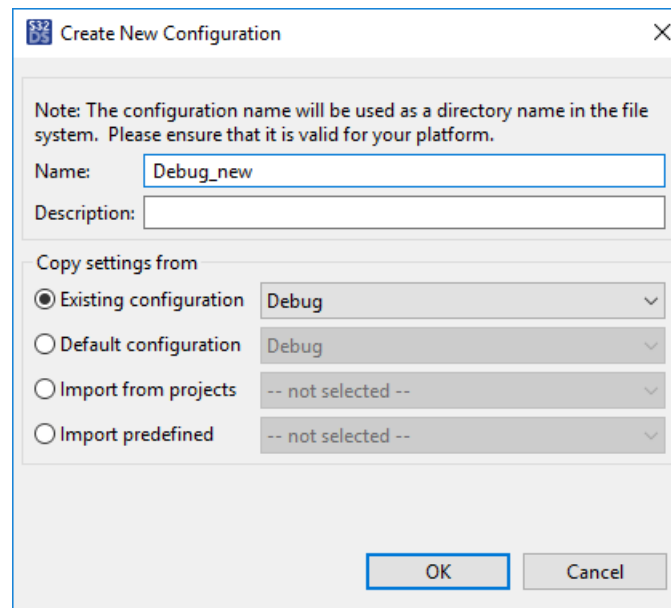
You can create as many build configurations for a project as required.

To create a new build configuration:

1. Click the project name in the **Project Explorer**, then click **Project > Build Configurations > Manage** on the main menu.
2. In the **Manage Configurations** dialog box, click **New**.



3. In the **Create New Configuration** dialog box, specify the name for the new configuration.



Select the original build configuration whose settings will be copied to the new build configuration:

- **Existing configuration:** Click this option to select a build configuration belonging to the project.

- **Default configuration:** Click this option to select a default build configuration generated for the project by project creation wizard.
- **Import from projects:** Click this option to select a build configuration belonging to any project available in the workspace.
- **Import predefined:** Click this option to select a build configuration predefined in a CDT toolchain.

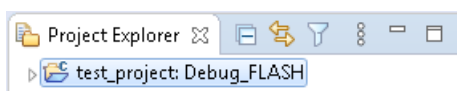
Then expand the drop-down menu next to the selected option and pick the build configuration.

4. Click **OK** and again **OK**.

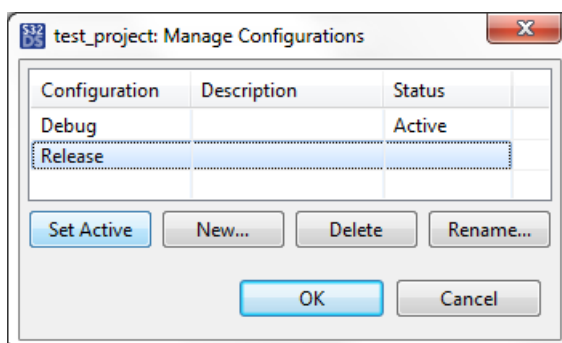
If required, edit the new build configuration settings as described in [Editing a build configuration](#).

Setting the active build configuration

One of the project's build configurations is active, that is, used by default in the project builds. The active build configuration appears in the **Project Explorer** next to the project name:



To select a different active configuration, open the **Manage Configurations** dialog box (**Project > Build configurations > Manage** on the menu), click the configuration to be active, and click the **Set Active** button.

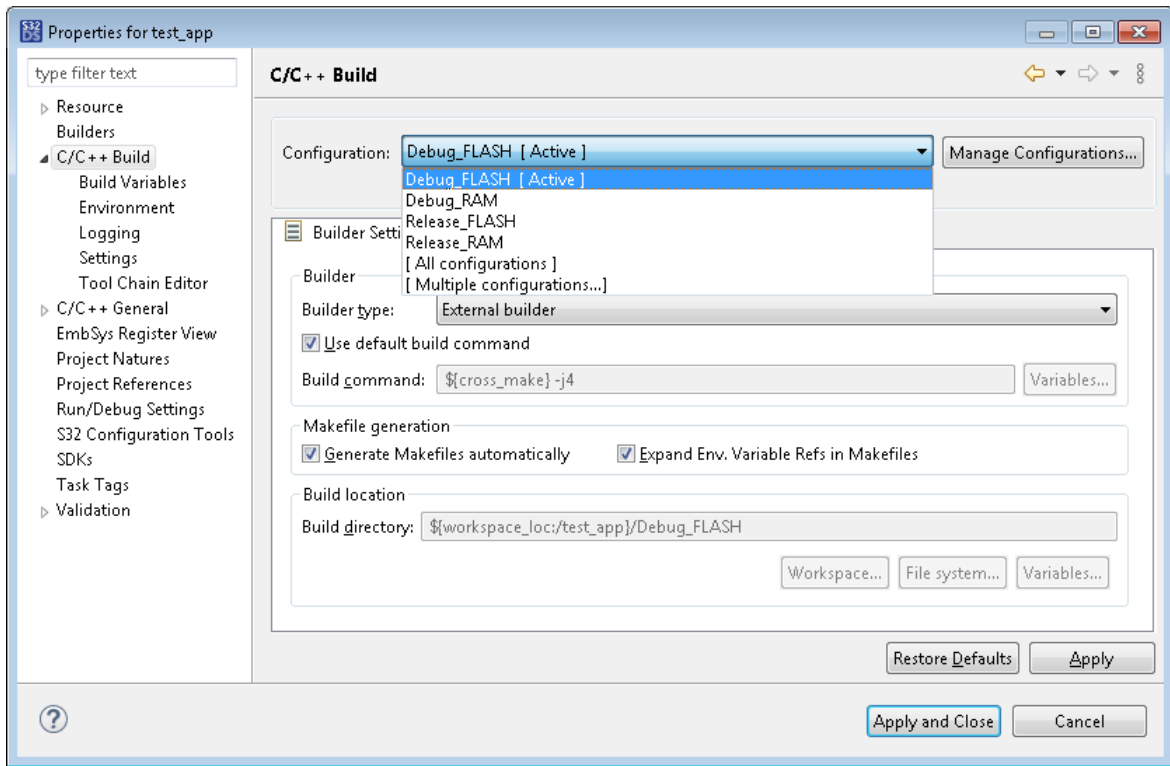


Another option is, right-click the project name in the **Project Explorer** and click **Build configurations > Set Active** and the required configuration from the context menu.

Editing a build configuration

To open a build configuration for editing, go to the project properties (**Project > Properties** on the menu) and then to the **C/C++ Build** section in the left pane.

Expand the **Configuration** menu to view all build configurations available for the project. Select the build configuration to be viewed and edited:



The build configuration settings are arranged in groups and are available from the **C/C++ Build** section in the left pane:

Table 3: Build configuration settings

Settings	Description	Refer to:
C/C++ Build page	Builder Settings tab: Click to select the builder type and to configure the builder settings.	Configuring the builder
	Behavior tab: Click to configure the build flow.	Configuring the build behavior
	Refresh Policy tab: Click to configure the list of resources to be refreshed after each build. Applies to the external builder only.	
Build Variables page	Click to view and edit the list of build variables to be used by the external builder for string substitution.	
Environment page	Click to view and edit the list of environment variables available to the builder.	
Logging page	Click to enable and configure build logging.	
Settings page	Tool Settings tab: Click to configure the tool settings.	Tool settings section (<i>Reference</i>)
	Build Steps tab: Click to add optional pre-build and post-build steps.	Adding the pre-build and post-build steps
Tool Chain Editor page	Click to select a different builder for the project. Also, you can select the tools in the current toolchain to be called at build time.	

Note: Click **Apply** to save the latest changes to the build configuration. If you just close the **Properties** dialog box, your updates will be lost.

Configuring the builder

To edit the builder settings, open the project properties and click **C/C++ Build** in the left pane and the **Builder Settings** tab in the right pane.

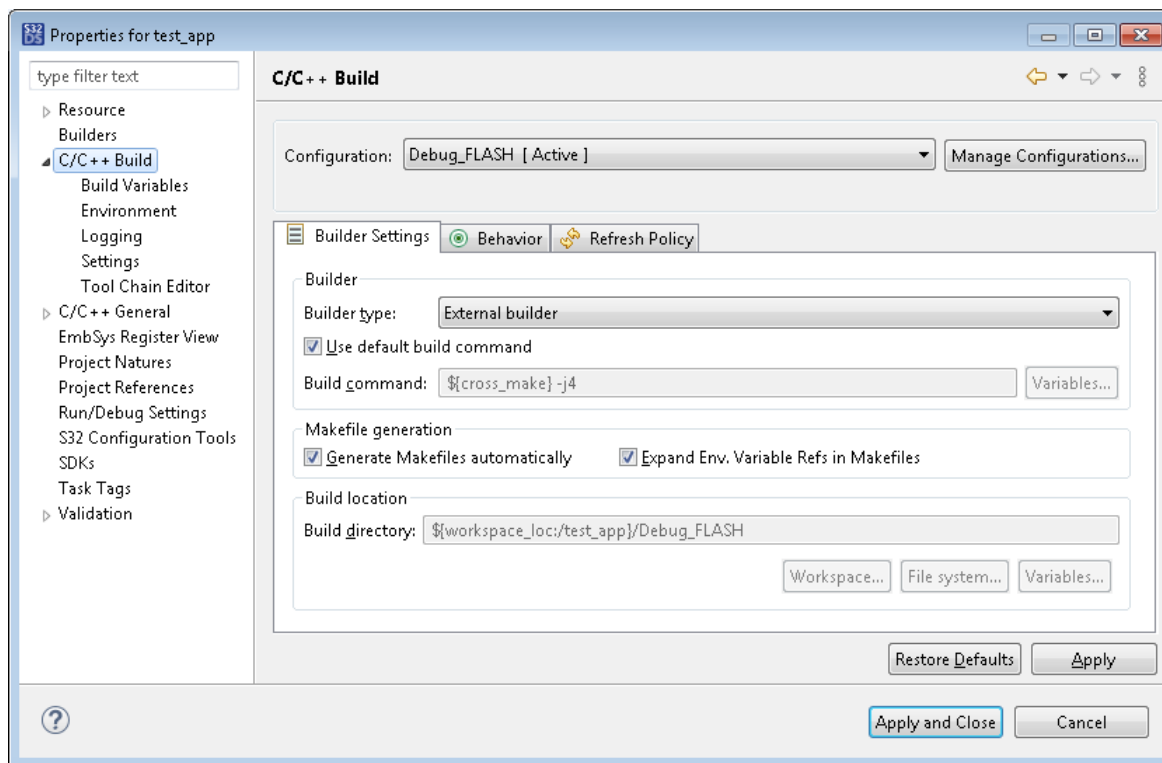


Table 4: C/C++ Build: Builder Settings

Settings	Description
Builder	Specify the builder settings: <ul style="list-style-type: none"> • Builder type: Select “External builder” if you need a makefile to be generated. Otherwise, select “Internal builder”, in which case all the remaining settings on the tab become disabled. • Use default build command: Enable this option to use the default build command (shown in the Build command field). Remove the flag to edit the build command. • Build command: Specify the build command.
Makefile generation	Generate makefiles automatically: Enable this option to generate a makefile with each build. Remove the flag if you have edited the makefile manually and need to keep these changes in the next builds. If required, specify a different folder for build output in the Build directory field.
Build location	Build directory: Specify the path of the build folder.

Configuring the build behavior

To edit the build behavior, open the project properties and click **C/C++ Build** in the left pane and the **Behavior** tab in the right pane.

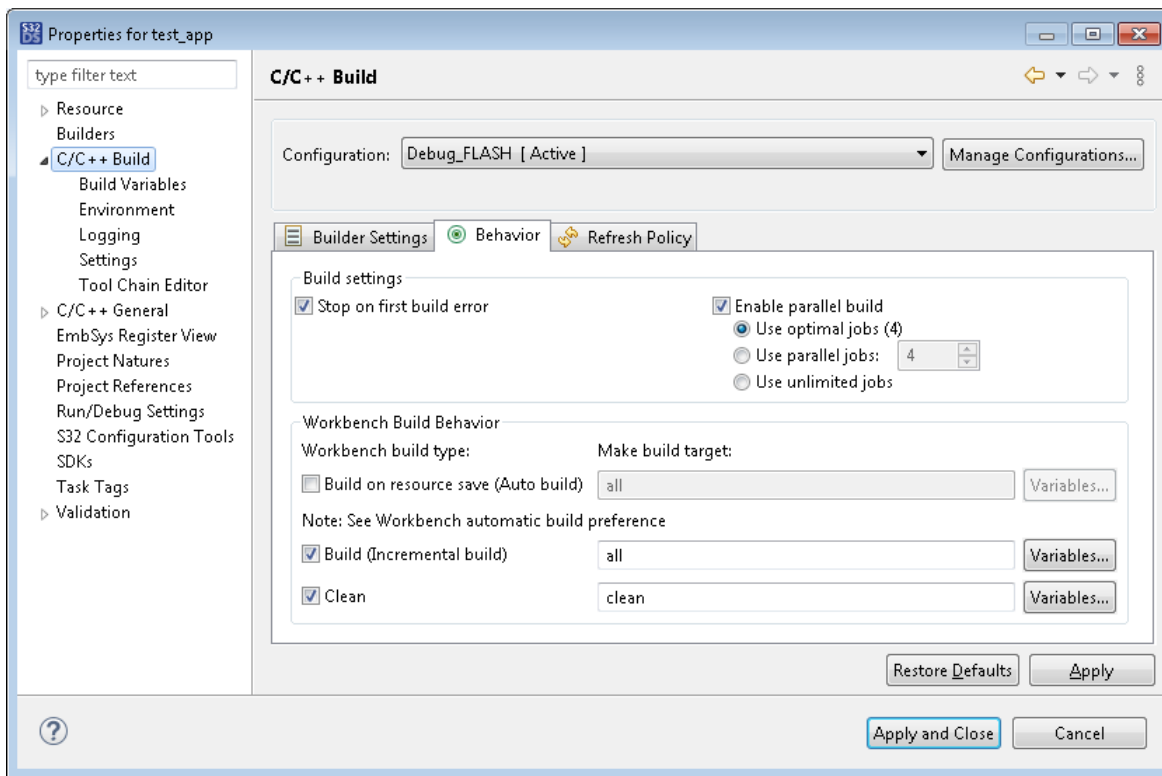


Table 5: C/C++ Build: Behavior settings

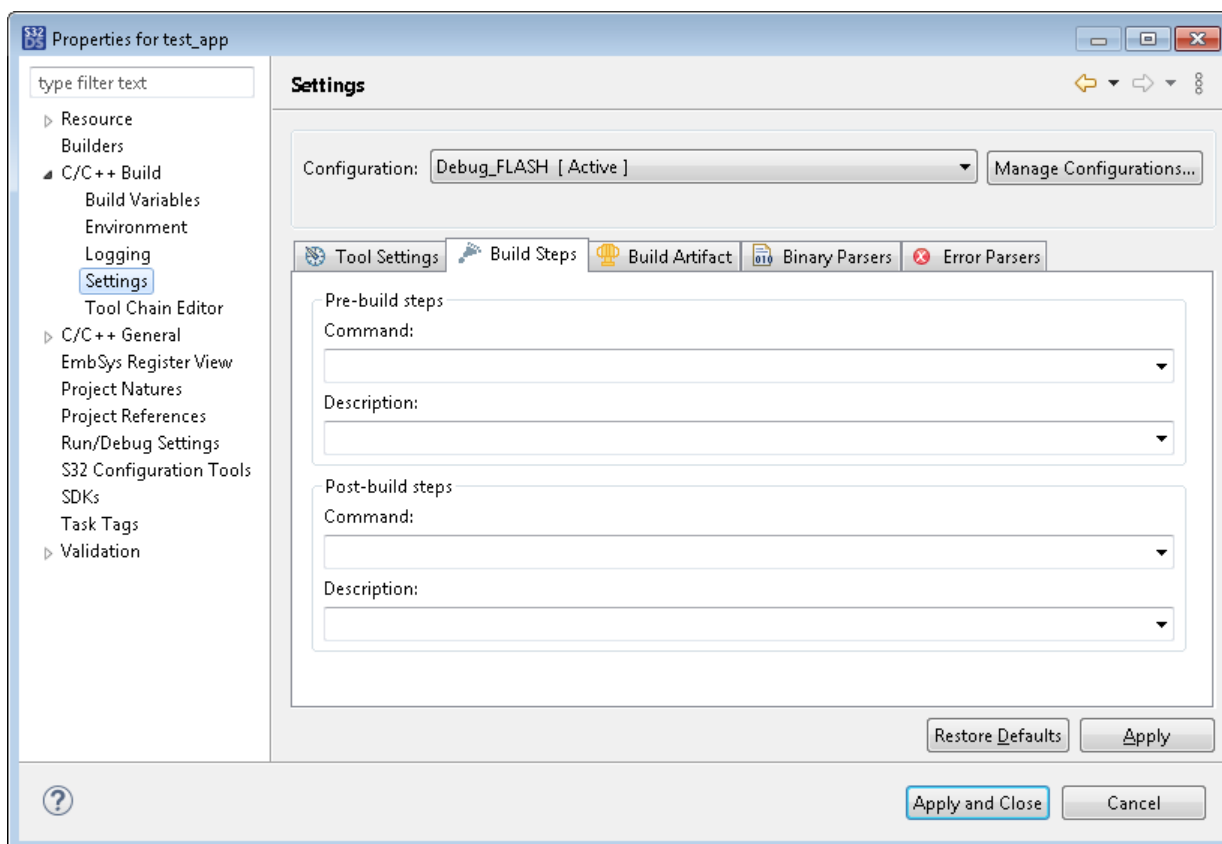
Settings	Description
Build settings	<p>Specify the build options:</p> <ul style="list-style-type: none"> • Stop on first build error: This option stops the build at the first encountered error. The option is enabled by default and read-only. • Enable parallel build: Keep this option enabled to speed up the compilation by using parallel jobs. Configure the number of parallel jobs: <ul style="list-style-type: none"> • Use optimal jobs (4): Use the default number of four jobs. • Use parallel jobs: Specify the number of parallel jobs by the number of CPU on your machine. • Use unlimited jobs: Enable the builder to produce as many jobs as required by the build.
Workbench Build Behavior	<p>Specify the settings to be used by the builder when instructed to build, rebuild, or clean the project.</p> <ul style="list-style-type: none"> • Build on resource save (Auto build): Enable this option to rebuild the project whenever the project resources are saved. Click Variables to specify the path of the build target. • Build (incremental build): Enable this option to create incremental builds. Click Variables to specify the path of the build target to be incremented. • Clean: Enable this option to clean the project before rebuilding it. Click Variables to specify the path of the build target.

Adding the pre-build and post-build steps

A build configuration can optionally execute pre-build steps and post-build steps. If you specify a pre-build command, it will be executed before the build tools are called. A post-build command will be executed right after the ELF file is created.

To add a post-build step to the build configuration:

1. In the project properties, go to **C/C++ Build > Settings** and open the **Build Steps** tab.
2. Select the build configuration in the **Configuration** field and specify the post-build command under **Post-build steps**:

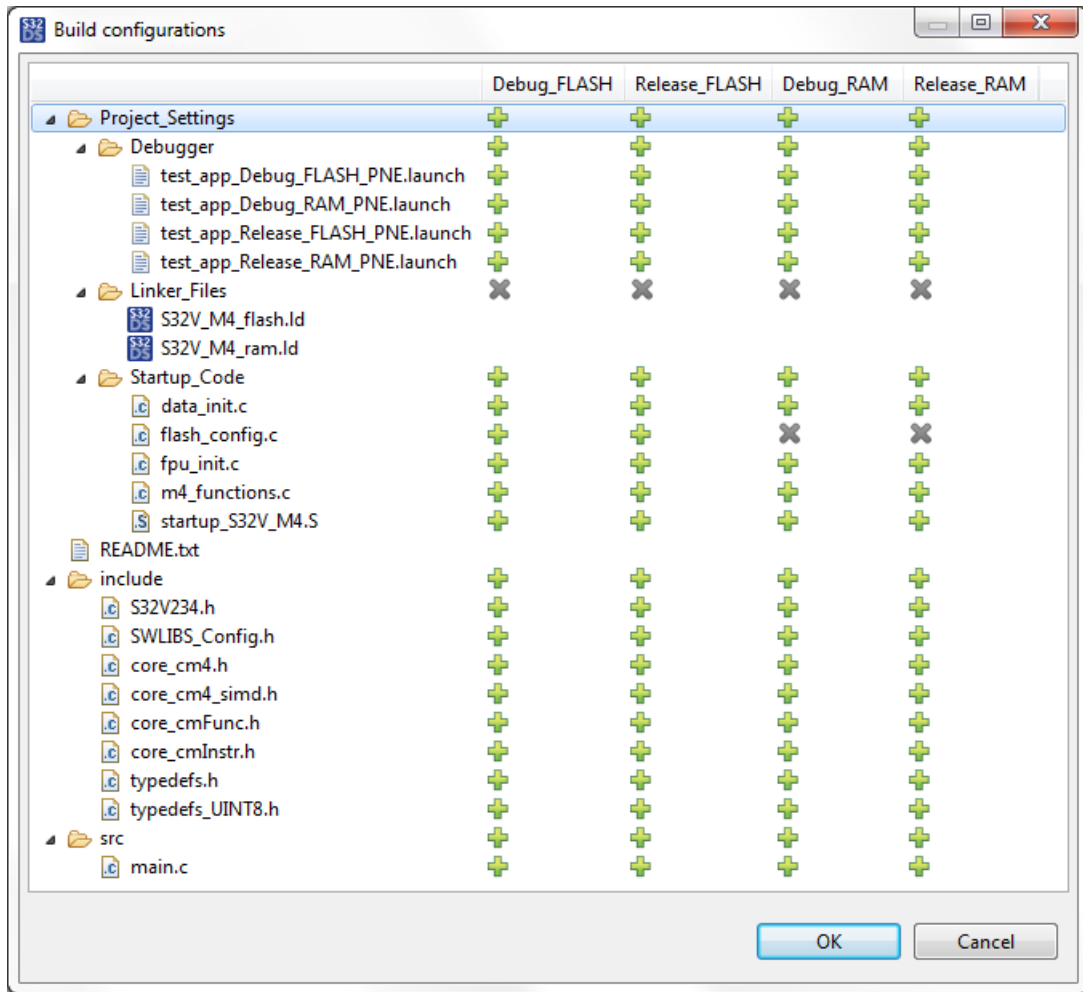


3. Click **Apply** and close the dialog box.
4. Build the project with the updated configuration. The **Console** view reports the execution of the post-build command.

Adding a pre-build step to a build configuration is similar, except you add your command under **Pre-build steps**.

Managing project resources in build configurations

To have a view of all project resources and of their use in each build configuration, right-click the project in the **Project Explorer** and click **Build Configurations Explorer** on the context menu.



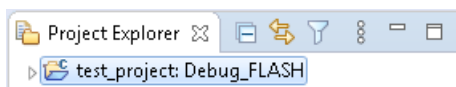
The **Build configurations** dialog box shows the data arranged in a pivot table. The tree of project folders and files is displayed in the left column of the table. The project's build configurations are represented by the columns located to the right from the tree.

If a particular folder or a file is included in a build configuration, the table shows the + sign at the intersection of the respective column and the row. An excluded resource is marked with the X sign.

You can modify the availability of the project resources in the **Build configurations** dialog box. To change the status of a certain resource to the opposite one, click the icon at the intersection of the respective column and the row.

Building a project

Before building a project, ensure that the project uses the right build configuration. The name of the active configuration is appended to the project name in the **Project Explorer** view:



If required, set a different active build configuration. Find the details in [Setting the active build configuration](#).

To build the project, click the project name in the **Project Explorer** view and do any of the following:

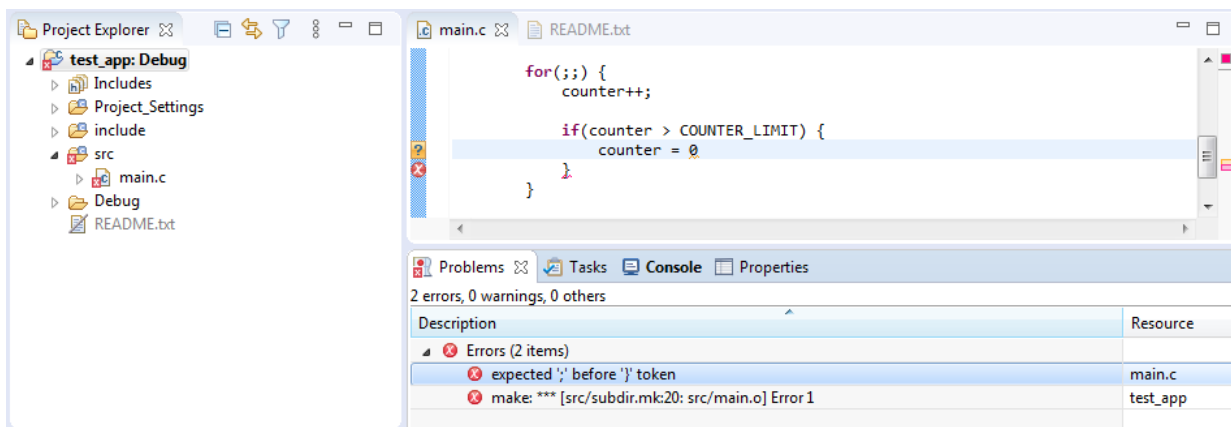
- Click the  (*Build*) button on the toolbar.

- Click **Project > Build Project** on the menu.
- Right-click the project name and click **Build Project** on the context menu.

When the build starts, all steps and the final status are reported in the **Console** view. To resolve possible build errors, refer to [Resolving build problems](#).

Resolving build problems

When the project build fails, the **Project Explorer** displays the “error” sign over the project's root folder and over the folders and files where the issues were detected. The **Problems** view reports about the issues that have caused the build failure:



Click an entry in the **Problems** view. If the issue is detected in code, the editor area displays the problem file and marks the line of code where the error is found.

Besides the syntax errors, build failures can be caused by invalid characters in the resolved paths. Such errors are reported without a particular location. If you get such an error in the **Problems** view, make sure that all of your variables hold paths with only ASCII characters ('A-Z', 'a-z'), digits ('0-9'), underscores ('_'), dash characters ('-'), periods ('.'), and slash characters ('/').

Note: Backslash ('\') is not allowed in paths.

Find an example of such an error in [Building projects in non-English versions of Windows](#).

Build failures with multiple unresolved symbols in code can be caused by the incorrect C/C++ indexer settings. Find the details in [Adjusting the C/C++ indexer settings for large files](#).

If system generates any warnings, you can also see them in the **Problems** view. You can use the **Quick Fix** option if possible:

- Right-click an error message and select **Quick Fix** option from the context menu,
- Select errors to fix,
- Click **Finish**.

After resolving all build issues, right-click the project in the **Project Explorer** and click **Clean Project** on the context menu.

Adjusting the C/C++ indexer settings for large files

The C/C++ indexer is an optional component that builds a database of project source and header files to provide C and C++ search, navigation features, and parts of content assist. If enabled, the C/C++ indexer updates the database every time you import, create, edit or delete a project file.

Indexing can be restricted by file size or by cache size, in which case the indexer may not update the database after some action was performed with a project file, for instance, because the file was too large. An attempt to build a project after that may fail with unresolved symbol claims, as displayed in the figure below:

```

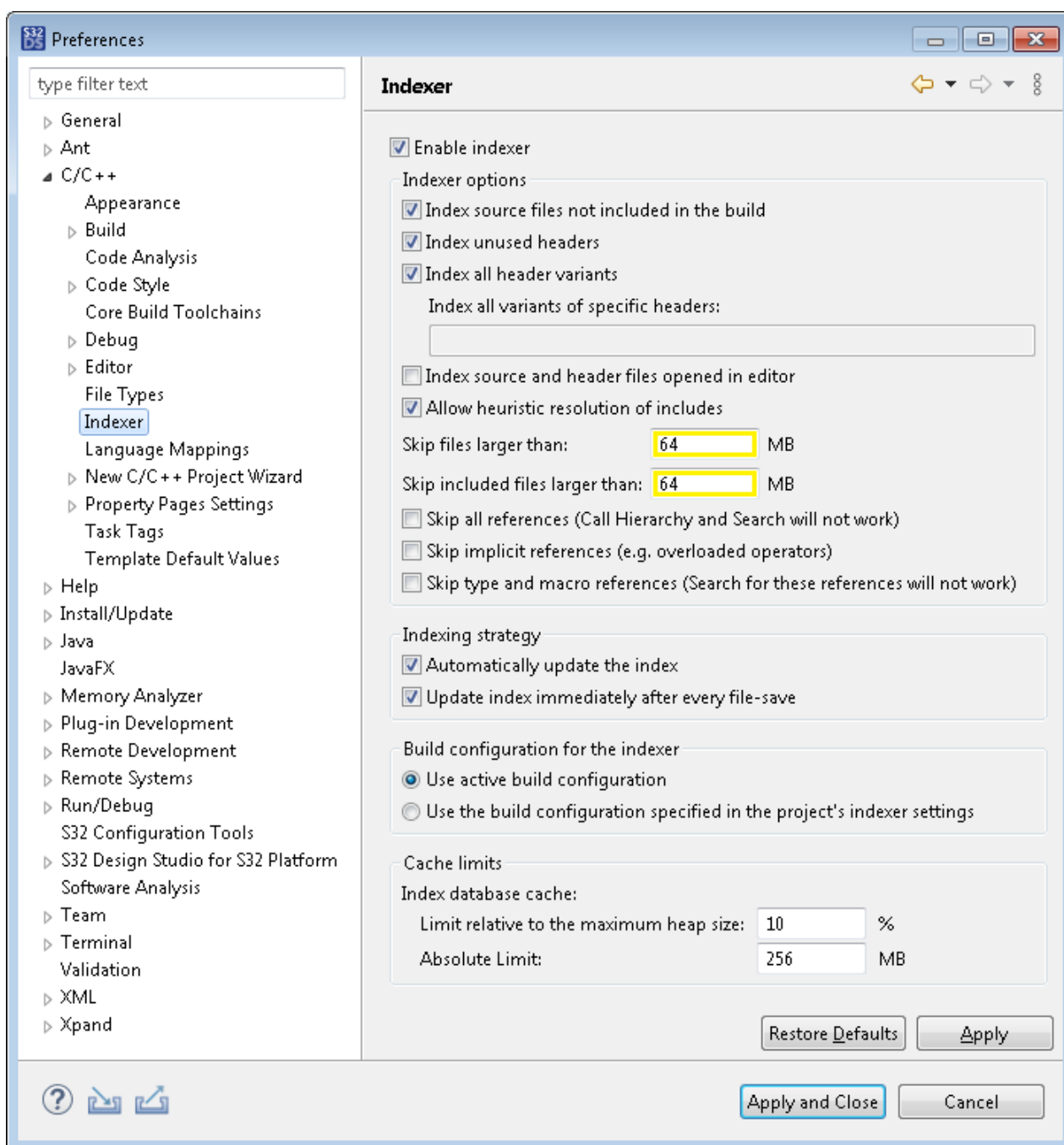
Base-CM7.vpcfg | S32R45.h | main.c | MCU_init.c | MCU_init.h
+@file      MCU_init.c
- /*****
  Includes
  *****/
#include "MCU_init.h"
#include "S32R45.h"
- /*****
  Global Functions
  *****/
- /** -----
  @brief   Initialise FXOSC to provide 40MHz internal clock
  @param   clock_mode_t clock_mode - Use either crystal or bypass mode
  @return  Success/failure indicator
  -----*/
- retval_t OSC_init(clock_mode_t clock_mode){
    uint32_t timeout = 0xFFFF;

    if(clock_mode == FXOSC_MODE){
        /* Set the CTRL register for functional mode: OSC_BYP = 0, COMP_EN = 1, EOCV = 0x9D, GM
        FXOSC.FXOSC_CTRL.R = 0x019D0011;
    }
    else if(clock_mode == FXOSC_BYPASS){
        /* Set the CTRL register for bypass mode: OSC_BYP = 1, COMP_EN = 0, EOCV = 0x9D, GM_SEL
        FXOSC.FXOSC_CTRL.R = 0x809D0011;
    }
    else return INVALID_ARGUMENT;

    /* Wait for FXOSC to stabilise */
    while((FXOSC.FXOSC_STAT.B.OSC_STAT != TRUE) && (timeout--));
    if(timeout < 1) return FXOSC_TIMEOUT;
    else return SUCCESS;
}
- /** -----
  @brief   Initialise the Core PLL
  @param
  @return  Success/failure indicator
  -----*/
- retval_t CORE_PLL_init(void){
  
```

In the above example, the project was imported from a Git server to an S32 Design Studio for S32 Platform workspace. The user was able to build the project without errors until the source file with the included large header file was opened. Now, at the attempt to rebuild the project the editor area flags errors on every single line that uses a symbol from that header file. The header file is included properly and is located at the include path, the compiler does not show any complains. Clicking an unresolved symbol with the *CTRL* key pressed leads straightly to the header file, but S32 Design Studio for S32 Platform still claims it cannot resolve the symbols.

The issue occurs because the size of the header file is more than 17 MB, while the C/C++ indexer is configured by default to skip files larger than 8 MB from indexing. To resolve the issue, click **Window > Preferences** and go to **C/C++ > Indexer**:



On the **Indexer** page, adjust the C/C++ indexer settings as follows:

- Enable the indexer (if disabled).
- Enable the **Index all header variants** option.
- Modify the **Skip files larger than** and **Skip included files larger than** settings so that no files in your project are skipped from indexing.
- Select **Use active build configuration**.
- Under **Cache limits**, raise the value of the **Absolute Limit** setting for the header file cache to 256 MB.

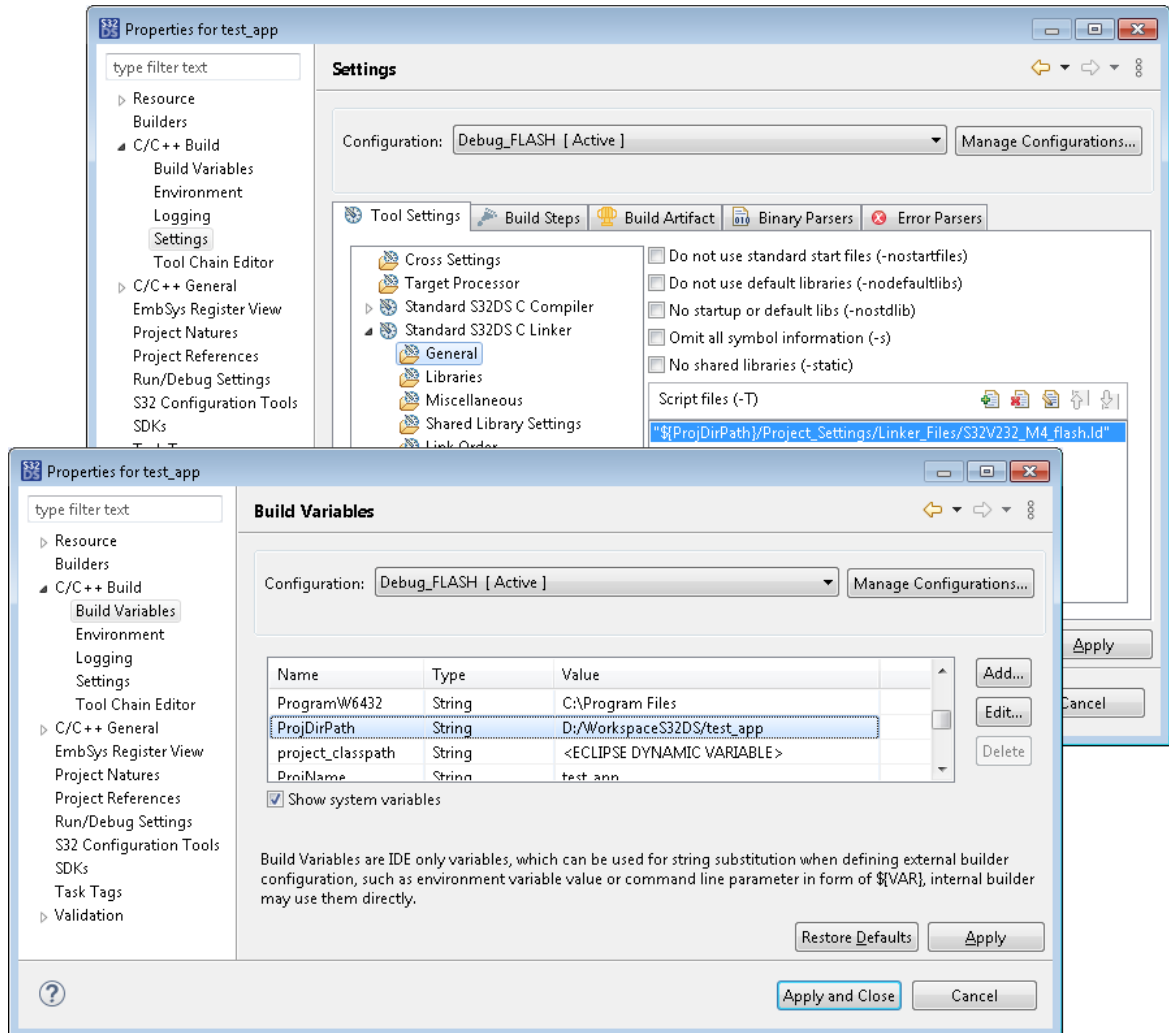
Click **Apply and Close**, clean and rebuild the project. If the errors still occur, restart S32 Design Studio for S32 Platform.

Note: The indexer settings adjusted in the **Preferences** dialog box apply to all projects in the workspace. To specify the indexer settings for a particular project only, open the project properties and go to **C/C++ General > Indexer**. Enable the project-specific settings and update the indexer settings as described above.

Building projects in non-English versions of Windows

Project builds may fail on a non-English version of Windows. The error occurs if the path to the linker file includes non-ASCII characters. The allowed characters are: A-z, 0-9, '-' (dash), '_' (underscore), '/' (slash), '.' (period).

Open the project properties and go to the **C/C++ Build > Settings > Tool Settings > Standard S32DS C/C++ Linker > General** page. In the **Script files (-T)** field, ensure that the path to the linker file uses the allowed characters and that all included variables are resolved to ASCII strings:



If necessary, relocate the linker file to a folder whose path includes the allowed characters, or rename the folders properly.

If the issue is caused by a variable, consider using the relative path that excludes the use of the variable. For example, the **Script files (-T)** field contains the following path:

```
`${ProjDirPath}/Project_Settings/Linker_Files/<project_name>.ld
```

If the ``${ProjDirPath}` variable holds the string with invalid characters, replace the value in the **Script files (-T)** field with the relative path that does not use the ``${ProjDirPath}` variable:

```
../Project_Settings/Linker_Files/<project_name>.ld
```

The update takes effect after you click **Apply**.

If the errors still occur, go to the **C/C++ Build > Settings > Tool Settings > Standard S32DS C/C++ Compiler > Includes** page and replace "`${ProjDirPath}/include`" with "`../include`".

Using optional tools

A build configuration can include optional tools such as:

- [Standard S32DS Create Flash Image](#): This tool generates a flash image of the build target.
- [Standard S32DS Create Listing](#): This tool generates the disassembly listing of the build target.
- [Standard S32DS Print Size](#): This tool prints the size of the produced application to the console.

To be executed at build time, these tools need to be enabled and set up in the build configuration. Find an example in topic [Generating an image file](#).

Also, a build configuration includes tools that are never called at build time:

- Standard S32DS C/C++ Preprocessor
- Standard S32DS Disassembler

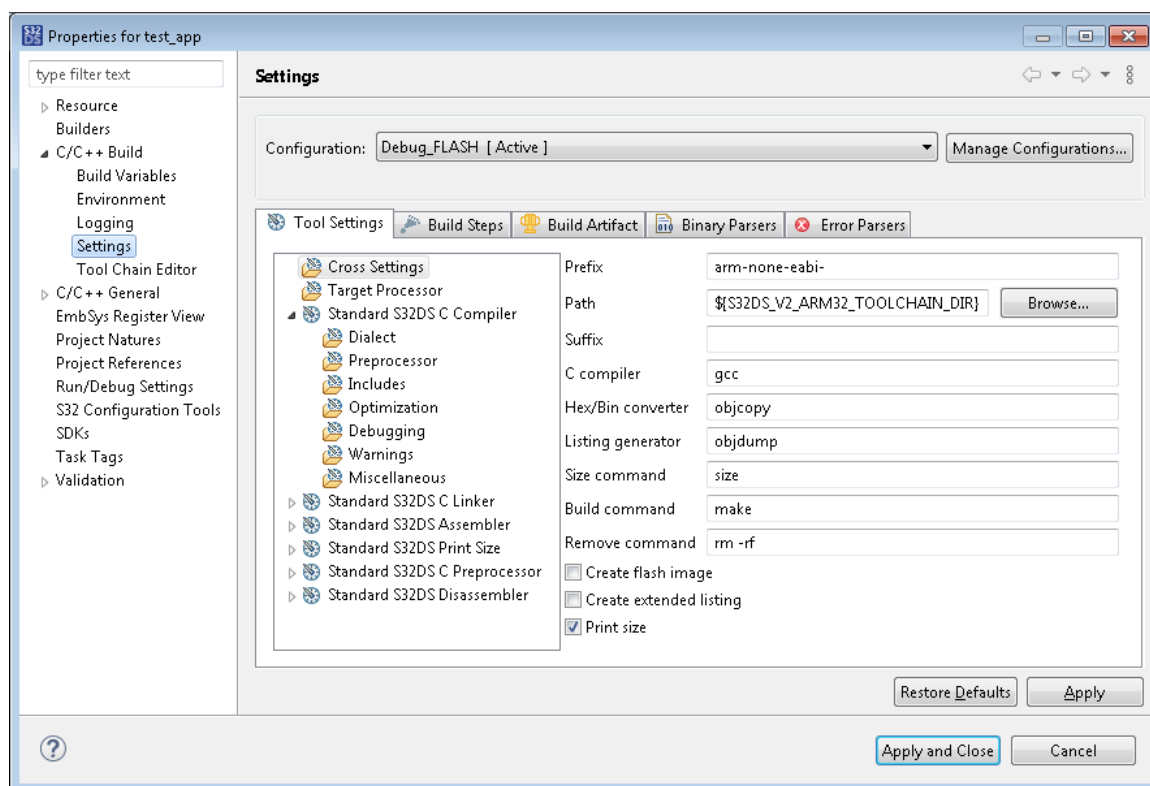
You can call these tools from the menu for the selected project files. Find the details in topics [Preprocessing source files](#) and [Disassembling binaries and source files](#).

Generating an image file

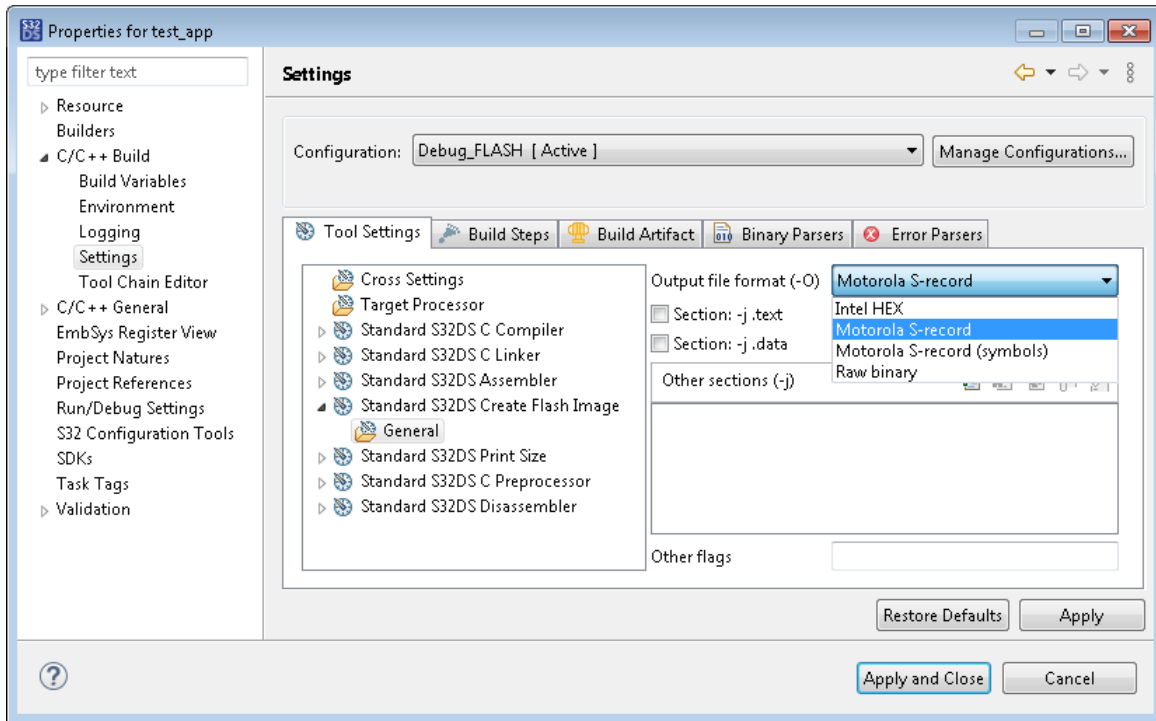
When building a project, you can optionally generate a flash image from the executable file.

To generate an image file:

1. In the project properties, go to **C/C++ Build > Settings > Cross Settings** and enable the **Create flash image** option.



- Click **Apply**. The left pane of the dialog box now shows the **Standard S32DS Create Flash Image** section.
- In the **Settings** page, go to **Standard S32DS Create Flash Image > General**. Select the required image format and specify other options. Find the details in the [Standard S32DS Create Flash Image](#) section (*Reference*) of this documentation.



Click **Apply and Close**.

- Build the project.
- In the **Project Explorer**, open the folder with the build output (**Debug** or other) and find the required image file.

Using output of optional tools in post-build steps

When specified in a build configuration, the [post-build command](#) will be executed after generating the ELF file and before calling the optional tools (for example, generating the SREC file). For this reason, when you try to use a file produced by an optional tool in a post-build command, you get the error message “No such file or directory”.

To use secondary outputs in a post-build command, you can do one of the following:

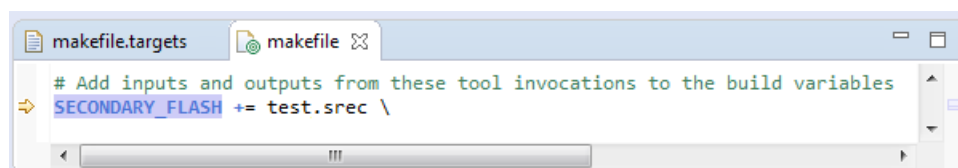
- [Move your post-build commands to the custom makefile](#). It will build your project as usual and then execute the post-build command right after the secondary output is produced.
- [Create and launch a sequence of two build configurations](#). The first configuration will produce secondary output (SREC, HEX and LST files), and the second configuration will execute the post-build commands for these files.

Using a custom makefile

- Create the `makefile.targets` file in the project root.
- Open this file in the editor and specify the target name, for example, `user_all`. It should build the default `all` target and then execute your post build commands.

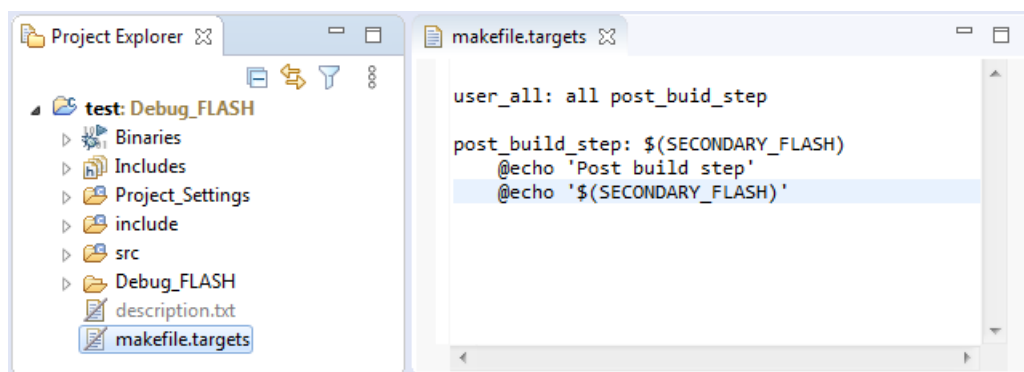
```
user_all: all post_build_step
```

- Create the `post_build_step` rule. It should be executed only if the optional tool output exists. In case of the SREC file, makefile defines the `SECONDARY_FLASH` variable for the output:

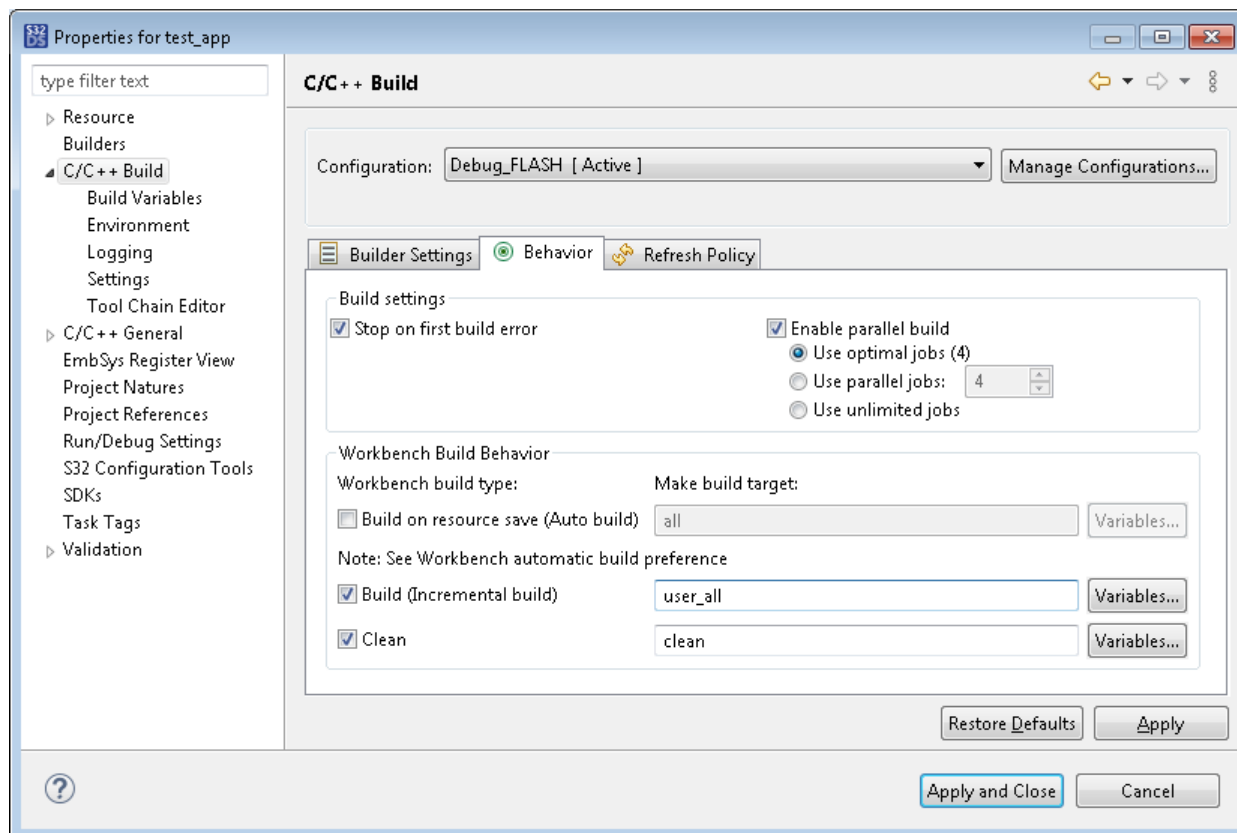


```
post_build_step: $(SECONDARY_FLASH)
```

4. Add the post-build commands.



5. In the project properties, click **C/C++ Build** and open the **Behavior** tab.
6. In the **Build (Incremental build)** field, specify your custom target instead of the default one.

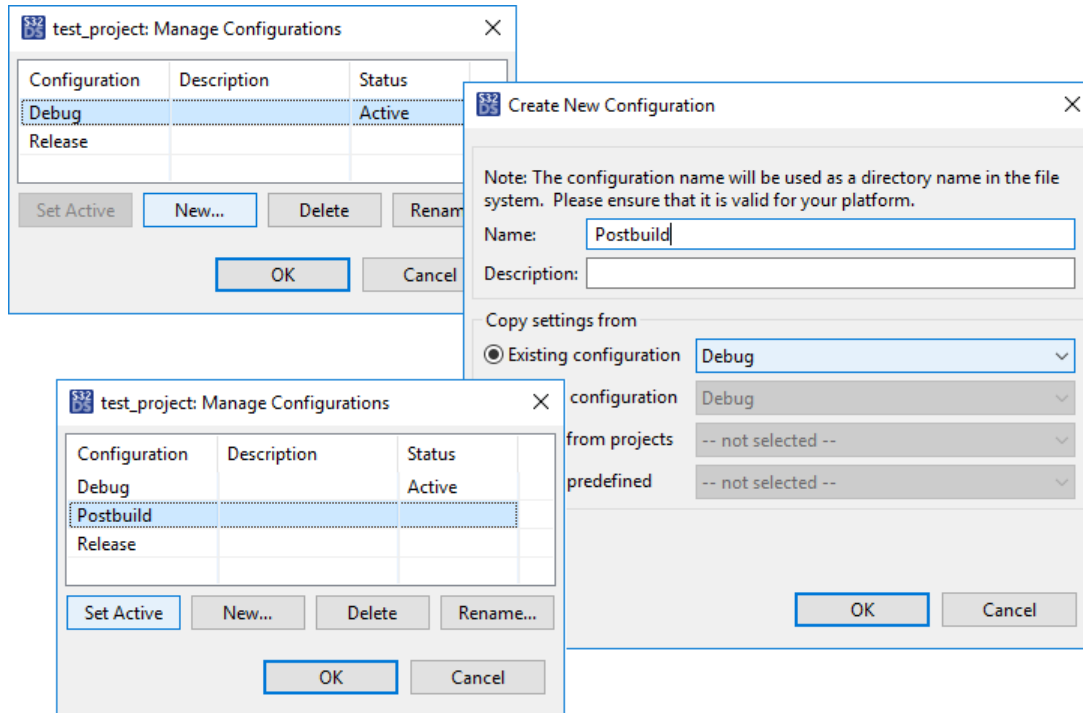


Using a sequence of two build configurations

1. In the project properties, go to **C/C++ Build > Settings**. Select the first build configuration, click **Cross Settings**, and select optional tools to be used for building the project.

For instance, select the **Create flash image** option to use the Standard S32DS Create Flash Image tool. Then go to **Standard S32DS Create Flash Image > General** and configure the tool.

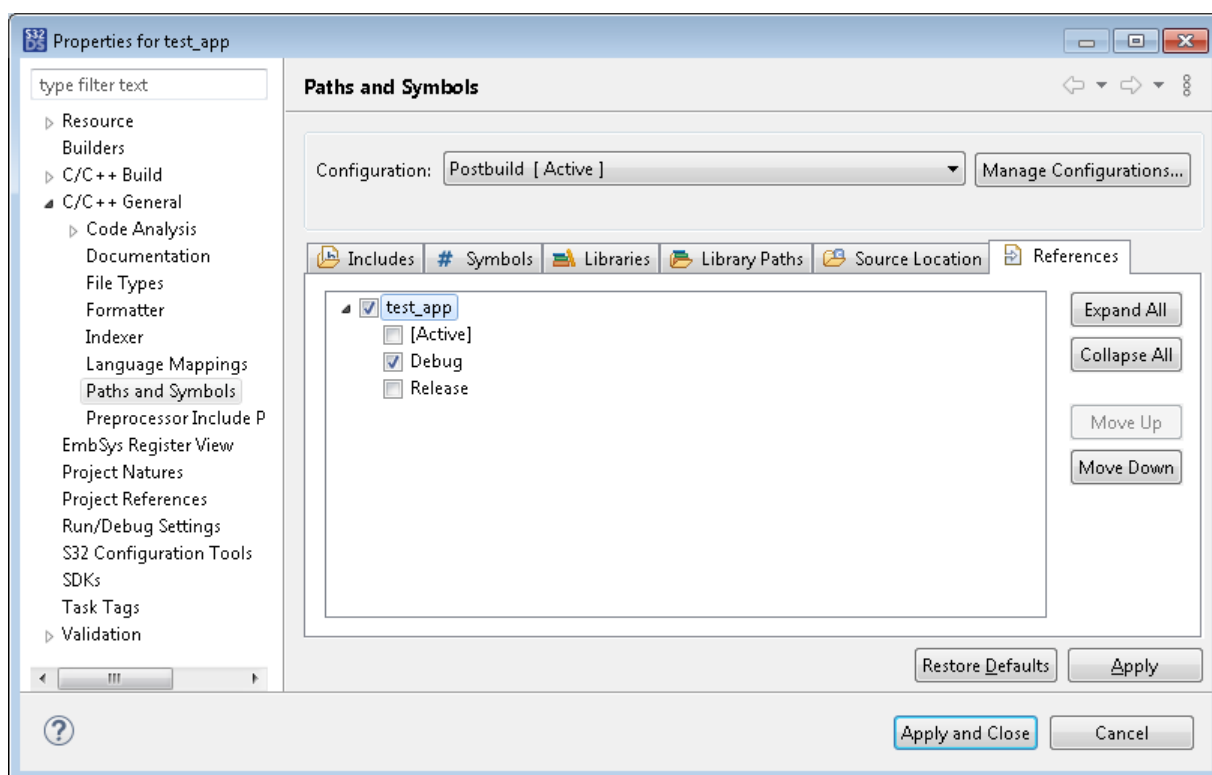
2. To create the second configuration, click **Manage Configurations** in the project properties, then click **New** in the pop-up dialog box. Specify the name of the second configuration. Choose to copy settings from the existing configuration and select the first configuration from the **Existing configuration** list. Click **OK**.



Then select the just created configuration in the pop-up dialog box and click **Set Active**. Click **OK**.

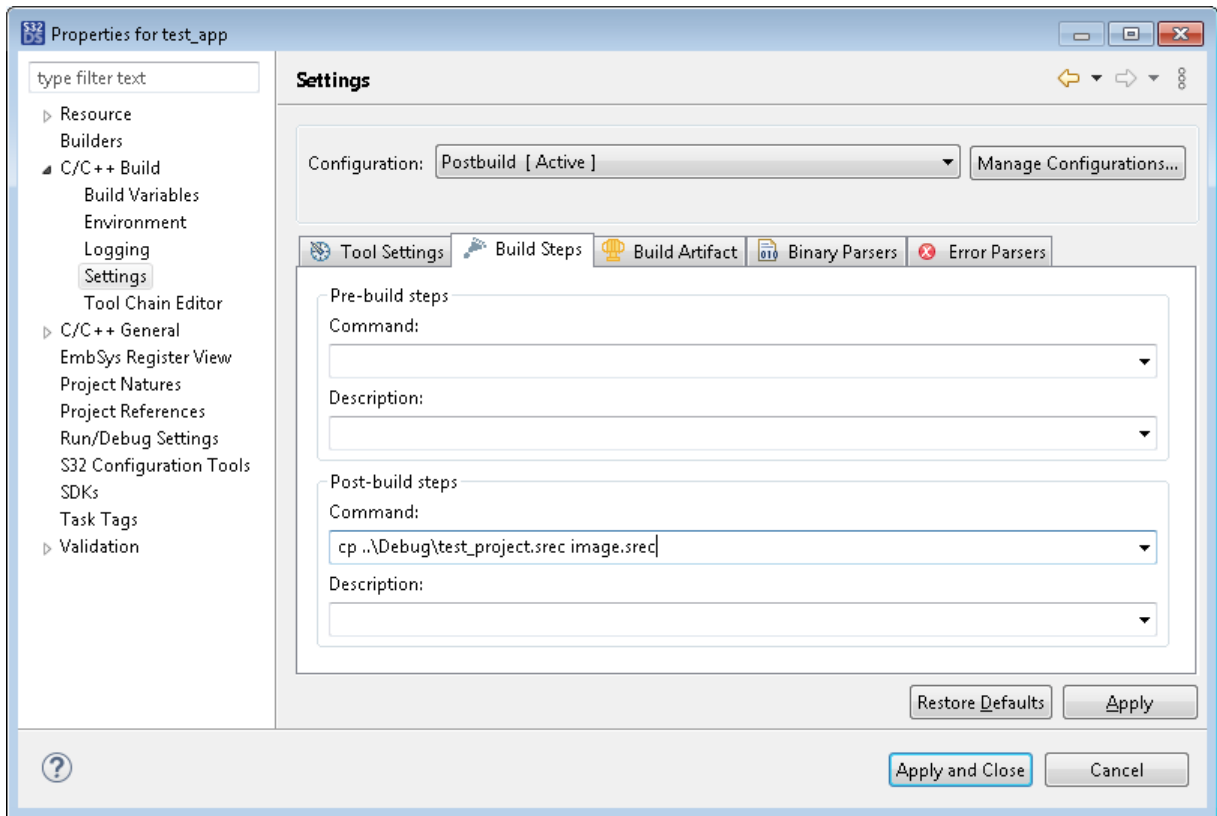
3. In the project properties, go to **C/C++ General > Path and Symbols**. Make sure the second configuration is selected in the **Configuration** field. On the **Reference** tab, expand the **<project_name>** list and flag the first configuration.

When you run the second configuration, this reference will call the first configuration prior to the second one.



- In the project properties, go to **C/C++ Build > Settings > Build Steps**. Select the second configuration in the **Configuration** field and specify the post-build command that uses secondary output created by the first configuration.

For instance, save a copy of the SREC file in the project folder where output of the second configuration is stored:



5. Run the second configuration.

This command launches the first build configuration that creates all specified build targets, including secondary output. Then the second configuration executes the post-build command.

Preprocessing source files

The Standard S32DS C/C++ Preprocessor tool enables you to evaluate the results of preprocessing of any C or CPP source file without starting the project build. The tool generates a text file in which all macros are resolved to their values.

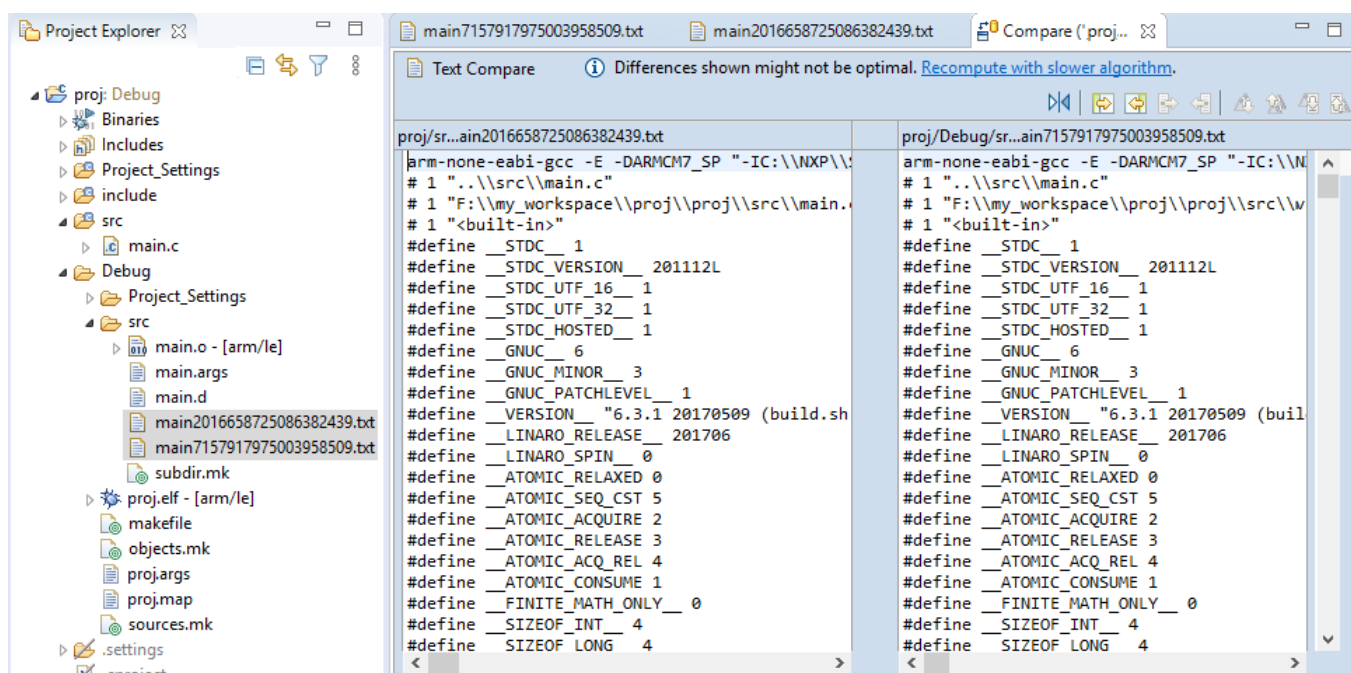
To preprocess a source file, do any of the following:

- In the **Project Explorer**, go to the **src** folder, right-click the C or CPP file and click **Preprocess Selected File(s)** on the context menu.
- Right-click the open source file in the editor area and click **Preprocess Selected File(s)** on the context menu.

Note: Files that are not included in the current build configuration cannot be preprocessed.

When done, the text file with the preprocessor output appears in the **Project Explorer** and in the editor area. If the resulting file exceeds 1 Mb in size, choose a file editor in the **Editor Selection** dialog box.

Each time the tool saves output to a new file with a unique name, allowing you to preprocess a source file many times, keeping the earlier results of preprocessing. To compare the output files, select them in the **Project Explorer**, right-click the selection, and click **Compare With > Each Other** on the context menu.



The tool is called with the “preprocess only” option (`-E`) by default. To enable additional options, refer to Standard S32DS C/C++ Preprocessor in the project properties (**C/C++ Build > Settings**).

Disassembling binaries and source files

The Standard S32DS Disassembler tool enables you to get disassembly of a binary or a source file without starting the project build. The disassembly code can be used to reveal the logical flaws or vulnerabilities.

To disassemble a file, do any of the following:

- In the **Project Explorer**, go to the **src** folder and right-click the C or CPP file, or right-click the binary file in the build output folder. Click **Disassemble Selected File(s)** on the context menu.
- Right-click the open source file in the editor area and click **Disassemble Selected File(s)** on the context menu.

Note: Files that are not included in the current build configuration cannot be disassembled.

When done, the text file with disassembly appears in the **Project Explorer** and in the editor area. If the resulting file exceeds 1 Mb in size, choose the file editor in the **Editor Selection** dialog box.

Each time the tool generates a file with a unique name, allowing you to disassemble different versions of the same project file and to compare the results. To compare two or more disassembly files, select them in the **Project Explorer**, right-click the selection, and click **Compare With > Each Other** on the context menu.

By default, the tool is configured to generate the following output:

- The assembler mnemonics for the machine instructions (`-d` option)
- The source code intermixed with disassembly (`-S` option)
- All available header information, including the symbol table and relocation entries (`-x` option)

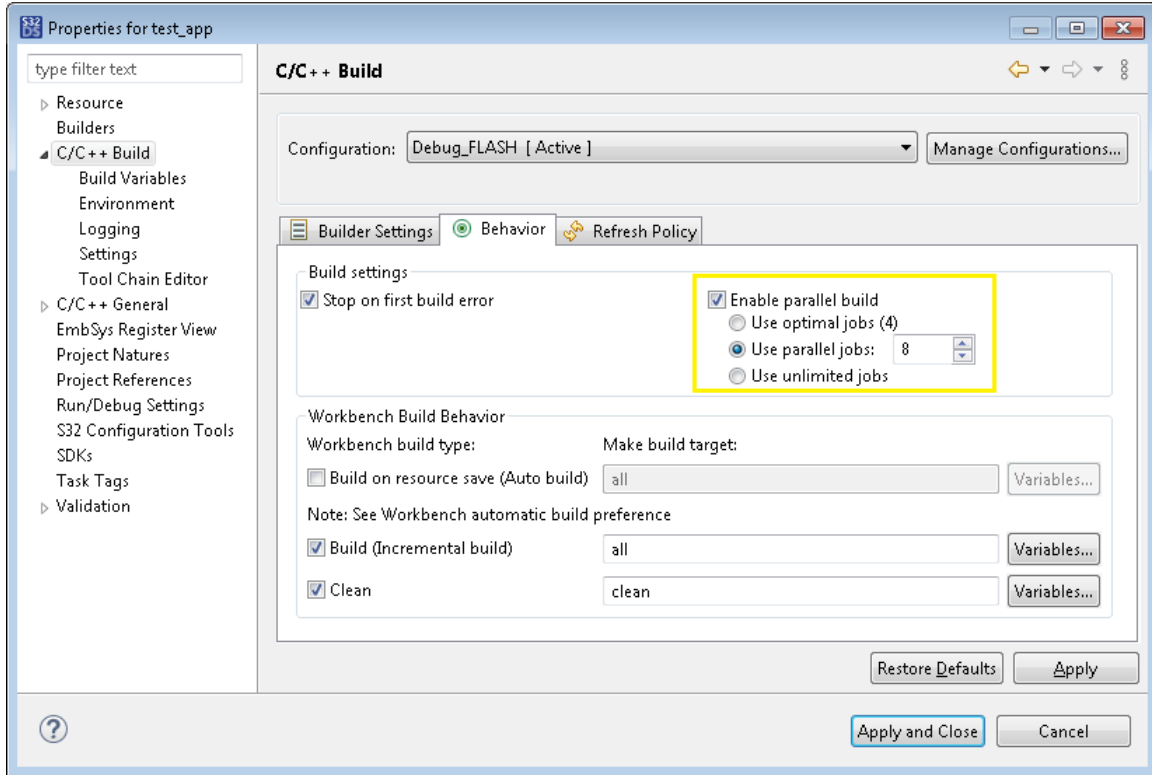
To enable additional options, refer to Standard S32DS Disassembler in the project properties (**C/C++ Build > Settings**).

Using parallel build

When you compile a large project, using the parallel build option can reduce the needed time by factors (typically, three to five times). The improvement depends on the number of cores and on the speed of your machine.

Parallel build means that the builder does not compile one file after each other, but instead runs a set of parallel builds. This is especially useful for host machines having multiple cores or CPU. Each CPU can do a compilation and balance the build load across all available CPU to cut the build time.

To speed up the build time, go to the project settings and click **C/C++Build** in the left pane. Go to the **Behavior** tab in the right pane:



The parallel build option is enabled by default. Select the number of “parallel jobs” depending on the number of CPU on your machine. Click **Apply** to save the changes and continue or **Apply and Close** to save the changes and quit the **Properties** dialog.

The possible disadvantage of the parallel build is, the compilation errors from all jobs are reported in the same **Problems** view in parallel. If the project has many errors in many files, consider turning parallel build off during the error fixing.

Debugging

Overview

One of the key features of S32 Design Studio for S32 Platform is its great support for debugging embedded C and C++ applications for S32 devices, both created in the wizard and imported to the workspace. Debugging is performed by means of the GDB tools supplemented with extra code (scripts, libraries, SDKs) to provide support for the target device, virtualization, and other options.

Support for each family of devices is brought to the debugging process with a respective software package or several packages that have to be installed into S32 Design Studio for S32 Platform.

Debugging environment

S32 Design Studio for S32 Platform allows debugging on a connected target, on a remote Linux system, and in the simulation mode. The availability of each option depends on the target device.

- To enable debugging on the target, S32 Design Studio for S32 Platform supports several evaluation boards with embedded devices as well as JTAG hardware debuggers from several manufacturers. S32 Design Studio for S32 Platform supports communication with a connected board via the USB and Ethernet interfaces.
- To enable debugging in the simulation mode, S32 Design Studio for S32 Platform implements integration with the Synopsys and VLAB tools. Debugging on a simulator is started and performed in special perspectives provided by S32 Design Studio for S32 Platform.

User interface

Debugging can be started and managed from the GDB command line and from the user graphical interface of S32 Design Studio for S32 Platform. When using the graphical interface, the developer is switched to the **Debug** or **VDK Debug** perspective that includes customizable views for inspecting values and managing the debugging process. The views available to the developer are **Memory**, **Registers**, **EmbSys Registers**, **Breakpoints**, **Expressions**, **Variables**, and other. The information about the debugging status appears in the **Console**, **Debugger Console**, and **Problems** views. The debug sessions with the included processes and threads can be managed in the **Debug** view.

Debug configuration

In S32 Design Studio for S32 Platform, a project is created with a collection of out-of-the-box debug configurations. A debug configuration consolidates all project specific debugging information. This includes the paths of the executable file and of the project, the settings responsible for the project rebuild, for connection and initialization of the board, for starting debuggers, and for file lookup. A debug configuration is the place to specify the first breakpoint and additional GDB commands. Optionally, a debug configuration can reference external symbols and an image file not included in the project.

The set of debug configurations created for a project includes at least one “debug” configuration and one “release” configuration. These configurations have similar settings and only differ in meta information. For projects created for debugging on a bare-metal target, S32 Design Studio for S32 Platform generates configurations for debugging a program from RAM and from flash.

Loading a program to the device

The simplest way to load a program to RAM or flash memory of the device is by running the respective debug configuration. The selected configuration already includes all settings and requires only a couple of final clicks from the developer.

While debugging from flash, the developer can use the command line interface to read flash memory and to reload a program to flash. This option is provided by the flash programmer tool for advanced debugging scenarios.

Another way to load a program directly to flash memory is by using the S32 Flash tool. This tool is installed with S32 Design Studio for S32 Platform but is run as a standalone application. Learn more in *S32 Flash Tool User Guide*.

Debugging

To start debugging, the developer runs one of the debug configurations available for the project. This event initializes the board and prepares the target cores (if applies), and executes the GDB commands specified in the configuration. Once done, a debug session is started. The GDB client and GDB server processes and the application process are executed in the context of the debug session.

If specified in the debug configuration, the debug session halts at the first breakpoint. The user releases the program execution and starts stepping through the code. During debugging, the user can set breakpoints, inspect and edit memory registers and peripheral registers, create variables and expressions, debug the source code or step through instructions.

Multi-core debugging

In S32 Design Studio for S32 Platform, the developer can start several concurrent debug sessions on different cores of the device and switch between them. Concurrent debug sessions can be started manually or using a launch group.

Launch group

S32 Design Studio for S32 Platform provides launch groups for running several executable files on a multi-core device with a single click. A launch group includes multiple debug configurations ranged the order of launching. When a developer runs a launch group, the included debug configurations are executed one after another.

A separate debug session is created for each launched configuration. The first (“boot”) debug session has to initialize the board, run and load the boot core, prepare the secondary cores, and start the GDB processes. The remaining (“secondary”) debug sessions load the code to the other cores. The debug sessions created by the launch group are managed and terminated independently.

Semihosting

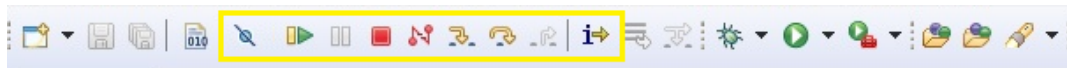
Semihosting is an option that allows user input and output of the debugging information in the console during debugging. Semihosting can be enabled in the debug configuration.

Code trace

Code trace collection is available for on-chip debugging with S32 Debug Probe. S32 Design Studio for S32 Platform integrates the tools for collecting code trace, for managing this process during debugging, and for analysis of the collected data. Learn more in *S32DS Software Analysis Documentation* available in the help system of S32 Design Studio for S32 Platform.

Using the debugger













The **Debug** perspective of S32 Design Studio for S32 Platform provides several toolbar buttons to let you manage active debug sessions. These buttons are available when a debug session is on.









Also, some debugging options are available from the editor area.

The following table describes all debugging options available to you when a debug session is on:

Table 6: Debugging options

Action	Button	Steps
Skip all breakpoints	 (<i>Skip All Breakpoints</i>)	Click Run > Skip All Breakpoints or click  (<i>Skip All Breakpoints</i>) on the toolbar.
Resume execution of the currently suspended thread	 (<i>Resume</i>)	Click Run > Resume or click  (<i>Resume</i>) on the toolbar.
Suspend execution of the currently selected thread	 (<i>Suspend</i>)	Click Run > Suspend or click  (<i>Suspend</i>) on the toolbar.
Stop execution of the currently selected debug session and/or process	 (<i>Terminate</i>)	Click Run > Terminate or click  (<i>Terminate</i>) on the toolbar.
Break the GDB connection	 (<i>Disconnect</i>)	Click Run > Disconnect or click  (<i>Disconnect</i>) on the toolbar.
Step into the routine call	 (<i>Step Into</i>)	Click Run > Step Into , or click  (<i>Step Into</i>) on the toolbar. The current statement is executed; then the current statement arrow moves to the next statement and stops. If the routine call is executed, then execution jumps to the first statement in that routine. If the last

Action	Button	Steps
		statement in the routine call is executed, then execution jumps to the next statement in the calling routine.
Step over the routine call	 (Step Over)	Click Run > Step Over , or click  (Step Over) on the toolbar. The current statement or routine executes; then program execution stops. If the current statement is a routine call, execution stops at the next breakpoint (watchpoint, eventpoint) or when the routine is finished.
Step out of the routine call	 (Step Return)	Click Run > Step Return , or click  (Step Return) on the toolbar. The rest of the current routine executes; then execution returns up the call chain and stops.
Enable/disable the Instruction Stepping mode	 (Instruction Stepping Mode)	Use the Instruction Stepping mode to step through instructions in the Disassembly view rather than through the source code in the editor. To enable/disable the Instruction Stepping mode, toggle the  (Instruction Stepping Mode) button on the toolbar.
Set a breakpoint (●) at the line of code		To halt execution of a statement, double-click the marker bar (the vertical ruler in the editor) in front of the statement. Or, right-click the marker bar and click Toggle Breakpoint . Note: The "No source file named..." warning appears in the Debugger Console view when you set breakpoints at similar function names in different projects. This is a known issue of Eclipse. This issue has no impact on the debug session and can be ignored.
Enable/disable a breakpoint		Right-click an active breakpoint (●) and click Disable Breakpoint from the context menu, or right-click a disabled breakpoint (○) and click Enable Breakpoint .
Remove a breakpoint		Double-click the breakpoint (● or ○).
Set the program counter		To continue execution from a certain line of code, right-click that line in the editor, then click Move To Line from the context menu. Note: Changing the program counter may cause your program to malfunction.
Restart execution of the program		Right-click the thread in the Debug view and click Relaunch from the context menu. Note: Relaunch is considerably faster to restart a debug session as it skips over loading debug information and register descriptors.

Using launch configurations

A launch configuration (or a debug configuration) is a collection of settings that describe how to launch a program. To run a program from S32 Design Studio for S32 Platform for debugging or for execution, you actually run a launch configuration with the respective settings.

A project can have as many launch configurations as required. When you create an application project in S32 Design Studio for S32 Platform, the wizard generates several launch configurations by default. Later on, you can create new launch configurations, delete the existing ones, and edit the configuration settings to better serve your needs.

The following topics describe how to manage launch configurations in S32 Design Studio for S32 Platform:

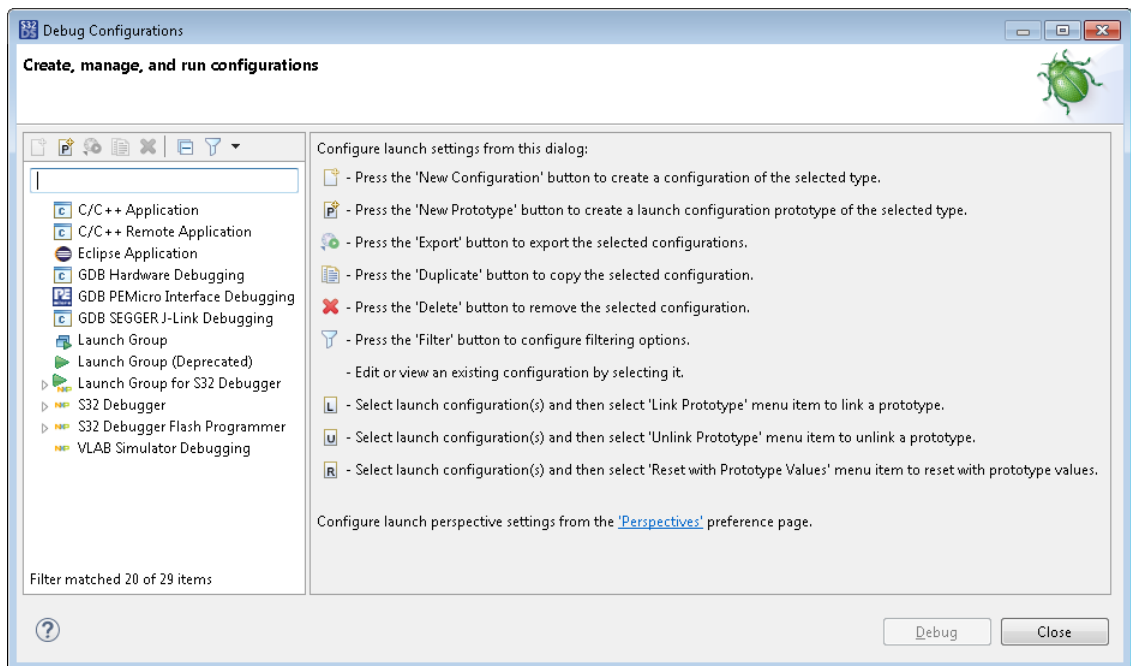
- [Creating a launch configuration](#)
- [Editing a launch configuration](#)
- [Running a launch configuration](#)

Note: Launching in Eclipse is closely tied to the infrastructure of debugging, so you can often meet the term “debug configuration” as well, meaning that the launch configuration is going to be used to interactively debug an application. Another term “run configuration” can be met in the context of the Run command applied to an application.

Creating a launch configuration

To create a launch configuration:

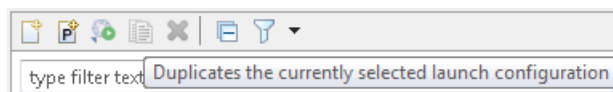
1. Right-click a project in the **Project Explorer**. Click **Debug As > Debug Configurations** on the context menu.
2. In the **Debug Configurations** dialog box, do any of the following:
 - In the left pane, click the debugging interface that fits your purpose:



- **C/C++ Remote Application:** Select for debugging on a remote Linux system.
- **GDB Hardware Debugging:** Select for debugging on Synopsys VDK (simulation).
- **GDB PEMicro Interface Debugging:** Select for debugging on a board connected to the PEMicro probe.
- **Lauterbach TRACE32 Debugger:** Select for debugging on a board connected to the Lauterbach probe.
- **S32 Debugger:** Select for debugging on a board connected to S32 Debug Probe.
- **S32 Debugger Flash Programmer:** Select for debugging on a board connected to S32 Debug Probe; the program is loaded to flash memory.
- **VLAB Simulator Debugging:** Select for debugging on VLAB (simulation).

Then click the **New launch configuration** toolbar button.

- In the left pane, double-click the required debugging interface and click a launch configuration inside. Then click the **Duplicate** toolbar button.



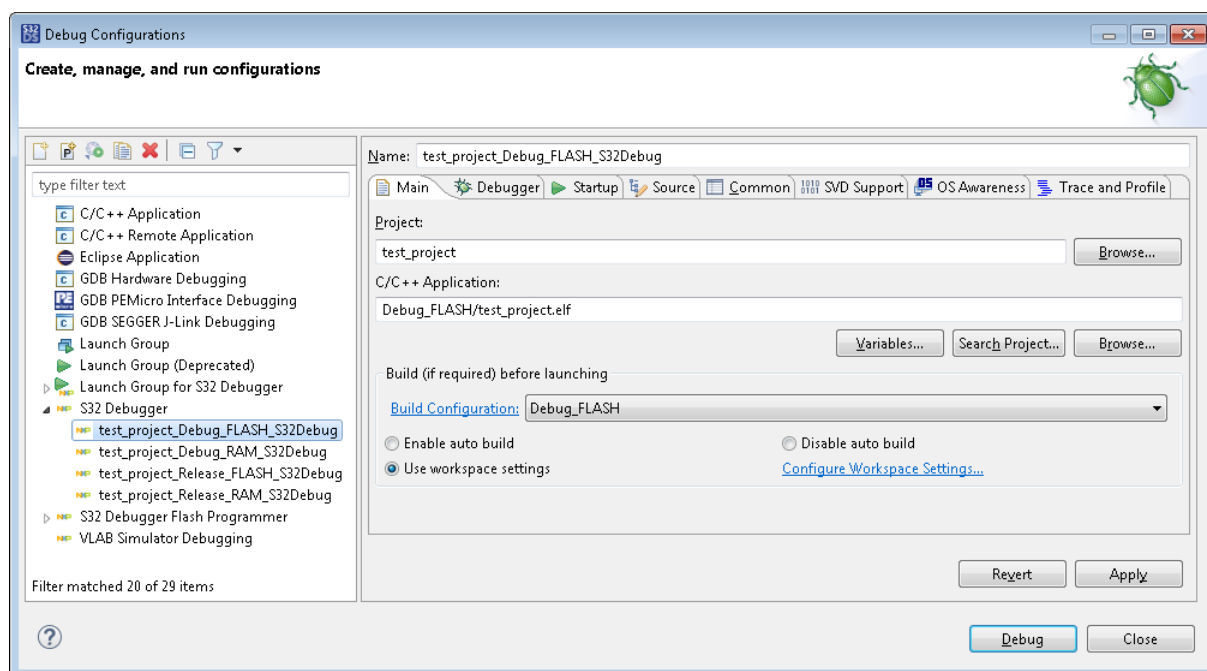
3. In the right pane, specify the name of the new configuration. The recommended format of a configuration name is *{project name}_{build configuration}_{target or debugger}*.
4. Specify the settings as described in topic [Editing a launch configuration](#). Click **Apply** to save the settings.
5. Click **Debug** to start debugging, or close the **Debug Configurations** dialog box.

Editing a launch configuration

To edit a launch configuration:

1. Open the launch configuration by right-clicking the project name in the **Project Explorer** and clicking **Debug As > Debug Configurations** on the context menu.
2. In the left pane of the **Debug Configurations** dialog box, expand the debugging interface specified in the project settings and click the required launch configuration.

For instance, if your project is created for debugging on a board using the S32 Debug Probe, expand the S32 Debugger interface and find the configuration starting with your project's name.



3. After you click the configuration in the left pane, the configuration settings appear in the right pane grouped in tabs. Open the required tab and modify the settings.

To learn about the settings in each group, refer to the respective topic:

- [Main tab](#)
- [Debugger tab](#)
- [Startup tab](#)
- [Source tab](#)
- [Common tab](#)
- [SVD Support tab](#)
- [OS Awareness tab](#)
- [Trace and Profile tab](#)

4. Click **Apply** to save the recent updates to the configuration. To discard the latest updates, click **Revert**.

Main tab

The **Main** tab of a launch configuration references the project and the executable file for debugging, and specifies the settings for automatic project builds.

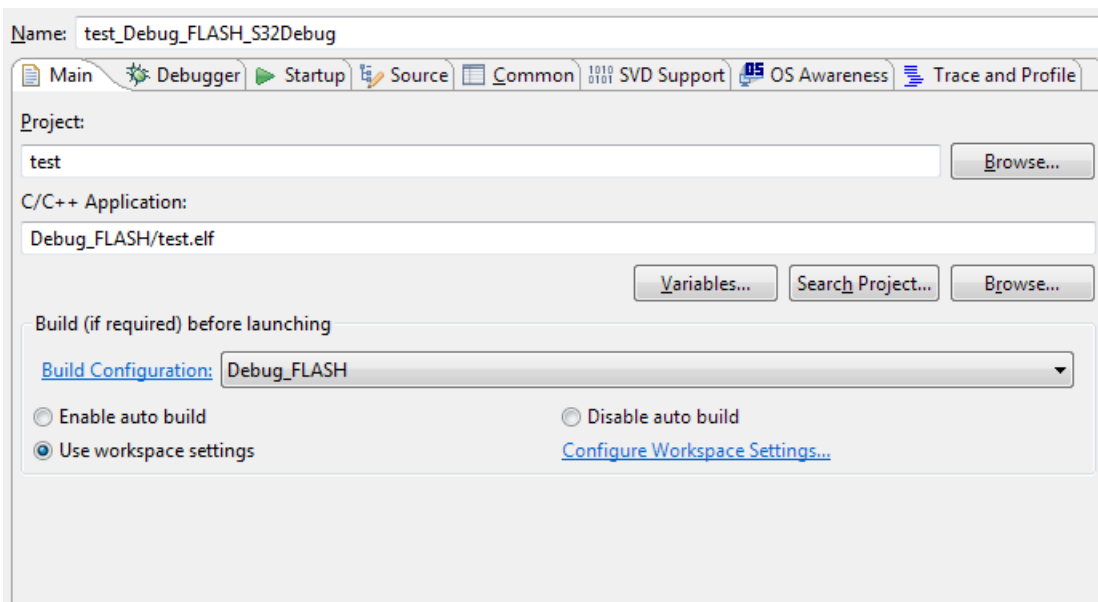


Table 7: Debug Configurations - Main tab (Generic settings)

Setting	Description
Project	Specify the project for which the launch configuration is created.
C/C++ Application	Specify the application (ELF file) to be launched.
Build (if required) before launching	Specify whether the application can be built automatically before launching. Select one of these options: <ul style="list-style-type: none"> • Build configuration: Specify the build configuration for automatic build. • Enable auto build: Enable automatic build if required. Selecting this option may cause the application to launch slower. • Disable auto build: Disable automatic build. • Use workspace settings: Use the option specified in the preferences of the current workspace. Click Configure Workspace Settings to view and edit the preferences.

If the launch configuration is specific for the C/C++ Remote Application debugging interface, the **Main** tab additionally configures the serial or TCP connection between the GDB host machine and the remote target:

Table 8: Debug Configurations - Main tab (C/C++ Remote Application settings)

Setting	Description
Connection	Select the connection between the target machine and the GDB host. Default: Local. If required, click New to create a new connection. Select the connection type (Serial Port, Telnet, or SSH) and specify the connection settings.

Setting	Description
Remote Absolute File Path for C/C++ Application	Specify the absolute path to the ELF file on the target machine.
Commands to execute before application	Specify commands to be executed on the target host before the debug session starts.
Skip download to target path	Enable this option to not download the program to the target machine.

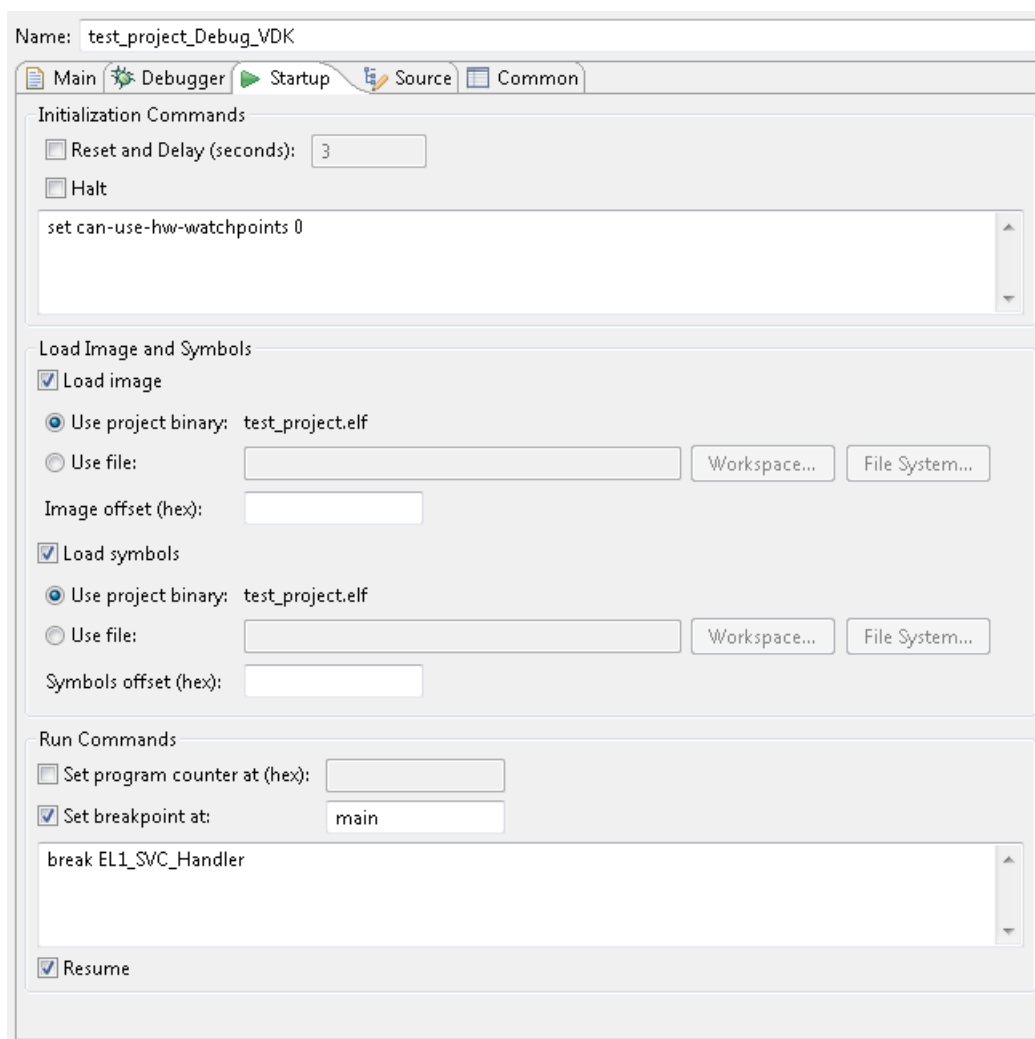
Debugger tab

The **Debugger** tab configures the debug session. The **Debugger** tab displays specific settings for debugging on a bare-metal target and on a Linux target, for debugging with a probe, and for debugging in the simulation environment. To learn about the required debugger settings in each case, refer to the following topics:

- [Debugging with S32 Debug Probe](#)
- [Debugging with S32 Debug Probe from flash for S32V23x targets](#)
- [Debugging with S32 Debug Probe from flash for other targets](#)
- [Debugging with a PEMicro probe](#)
- [Debugging with a Lauterbach probe](#)
- [Debugging on a Linux target](#)
- [Debugging on a VDK](#)

Startup tab

The **Startup** tab specifies the first commands for the GDB client to execute at startup. Commands are executed in the order of appearance on the tab, after which debugging of the code becomes available to the user.



The following table describes all settings that can appear on the **Startup** tab. Some settings described below are particular to the selected debugging interface and are hidden from the tab for other interfaces.

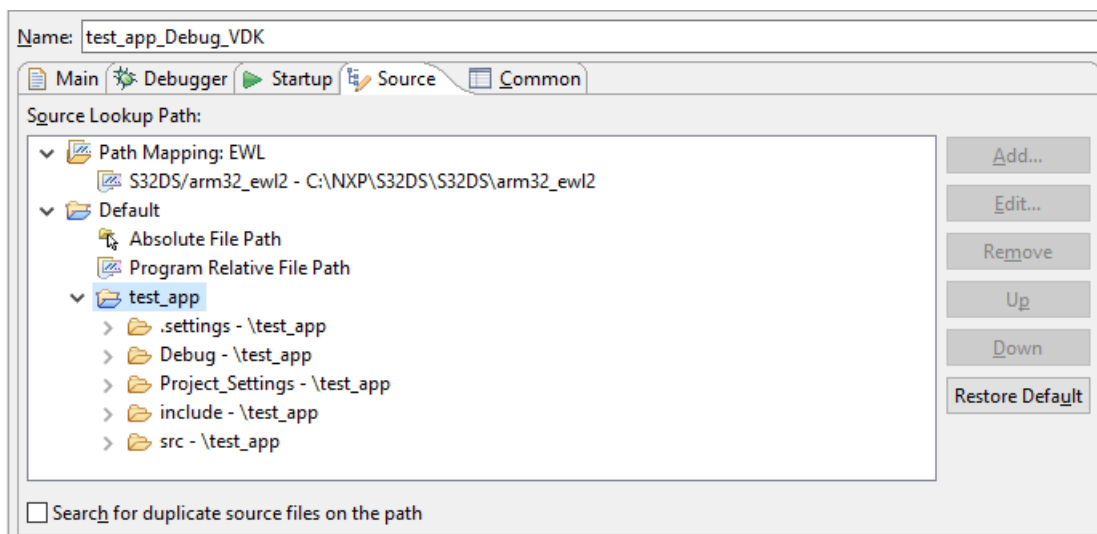
Table 9: Debug Configurations - Startup settings

Setting	Description
Initialization Commands (apply to GDB Hardware Debugging only)	Specify the initialization commands to be executed before the debug session is started. Configure the following related options: <ul style="list-style-type: none"> • Reset and Delay (seconds): Select this option to reset the target after programming and to delay the debug session. Specify the number of seconds for the delay. • Halt: Select this option to halt the target at startup.
Load Image and Symbols	Specify the file that contains the code for debugging: <ul style="list-style-type: none"> • Load image: Select this option to point the binary file for debugging. This file will be downloaded to the target: <ul style="list-style-type: none"> • Use project binary: Select this option to debug the project's executable. • Use file: Select this option to point a different ELF file for debugging. Click Workspace or File System to browse to the file.

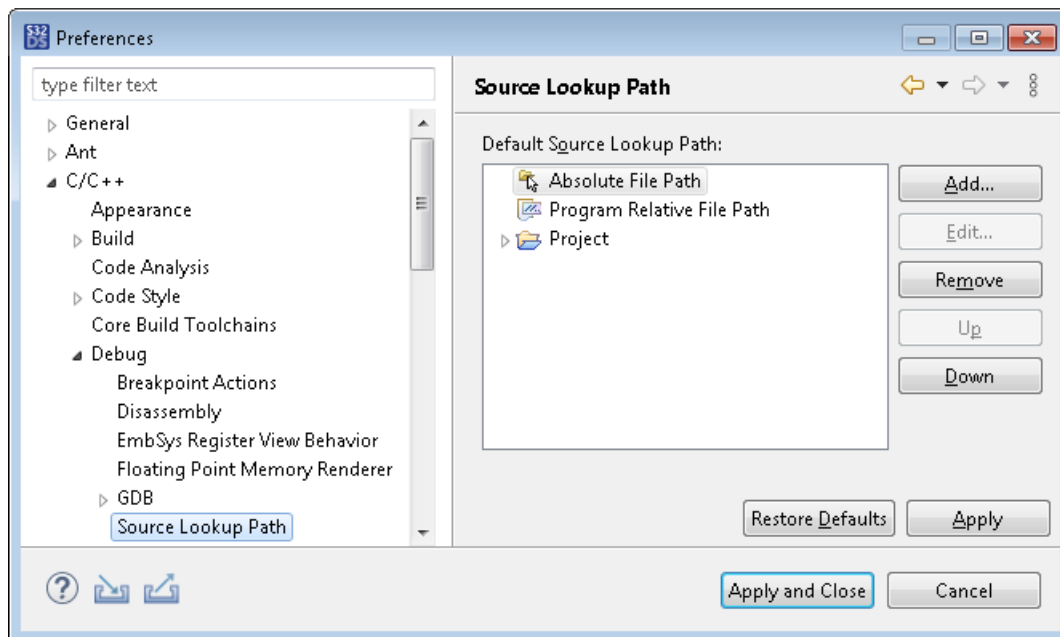
Setting	Description
	<ul style="list-style-type: none"> • Image offset (hex): Specify offset of the file section where the code starts. Leave this field empty if the code will be loaded from the project's executable. <p>Specify the source of the debugging information (“symbols”) to be passed to the debugger. The debugging information will be kept on the host rather than downloaded to the target.</p> <p>Typically, the debugging information (DWARF) is embedded in the ELF file generated from the project.</p> <ul style="list-style-type: none"> • Load symbols: Enable this option to point the file with symbols: <ul style="list-style-type: none"> • Use project binary: Select this option to use symbols from the project's executable. • Use file: Select this option to use symbols from a different ELF file. Click Workspace or File System to browse to the file. • Symbols offset (hex): Specify offset of the .text section in the file where the table of symbols starts. Leave this field empty if symbols will be fetched from the project's executable.
Runtime Options	<p>Specify the runtime settings for the debugger. Options:</p> <ul style="list-style-type: none"> • Set program counter at (hex): Select this option to set the program counter. • Set breakpoint at: Set the first breakpoint at the specified function. • Resume: Select this option to continue execution after the breakpoint.
Run Commands	<p>Commands to be executed by the GDB client after all above options are done. After that, the user can interact with the debugger.</p>

Source tab

The **Source** tab specifies the source lookup paths for the debugger.



A launch configuration created by the project creation wizard includes the default lookup paths. These default paths are specified in the preferences (**Window > Preferences > C/C++ > Debug > Source Lookup Path**) and apply to all projects created in the given workspace:



To modify the paths and their order in the scope of a given launch configuration, use the buttons located at the right side. When defining and prioritizing the file paths in the list, comply with the algorithm that the GDB debugger applies to find source files:

- The debugger starts looking for the file in the mapped directory (if any).
- If failed to find a readable file, the debugger searches the absolute path.
- If the previous step failed, the debugger searches the relative paths (program and project ones).

If the debugger fails to locate a readable file in all above paths, the debug session shows the “No source found for file” error message. To continue debugging, the user needs to locate the file manually.

By default, the debugger uses the first-found readable file as the source file. For the debugger to continue search on the path, select the **Search for duplicate source files ...** option.

Common tab

The **Common** tab allows you to specify the location of your configuration, input and output options, and launch options.

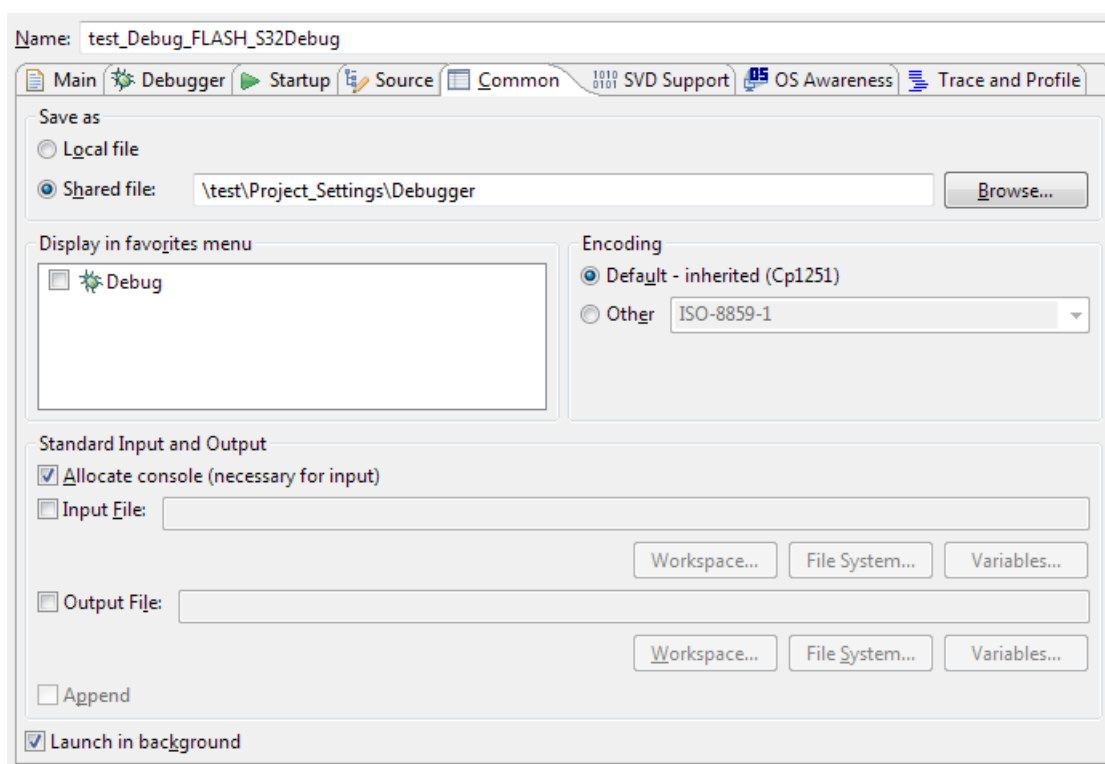


Table 10: Debug Configurations - Common settings

Setting	Description
Save as	Specify where to save the launch configuration you are editing. Select the required option: <ul style="list-style-type: none"> • Local file: Select this option to store the launch configuration file in a local system folder. • Shared file: Select this option to store the launch configuration file in the project structure. Click Browse and browse to the exact location. The LAUNCH file will appear in the specified project folder in the Project Explorer.
Display in favorites menu	Select the menu (Debug or Run , or both) in which your launch configuration will be displayed as the menu option.
Encoding	Select the encoding for output to be displayed in the Console view.
Standard Input and Output	Specify whether the debugger can receive input and display and save output. Select the required settings: <ul style="list-style-type: none"> • Allocate console (necessary for input): Select this option for the Console view to receive output for the debug session. • Input File: Select this option for the debugger to read input from a TXT file. Click Workspace or File System and browse to the file. • Output File: Select this option to save output from the Console view to a TXT file. Click Workspace or File System and browse to the file. • Append: If saving to the output file is selected, enable this option to append output to the end of the existing file rather than to rewrite the file.

Setting	Description
	<ul style="list-style-type: none"> • Launch in background: Select this option to launch the configuration in the background mode.

SVD Support tab

The **SVD Support** tab allows you to configure SVD files. This tab appears in debug configurations intended for debugging with PEMicro and S32 Debug Probes.

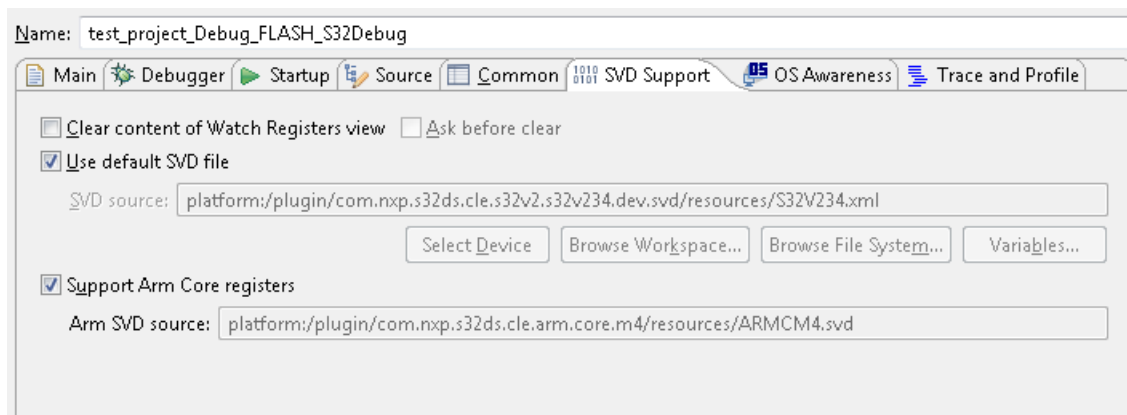
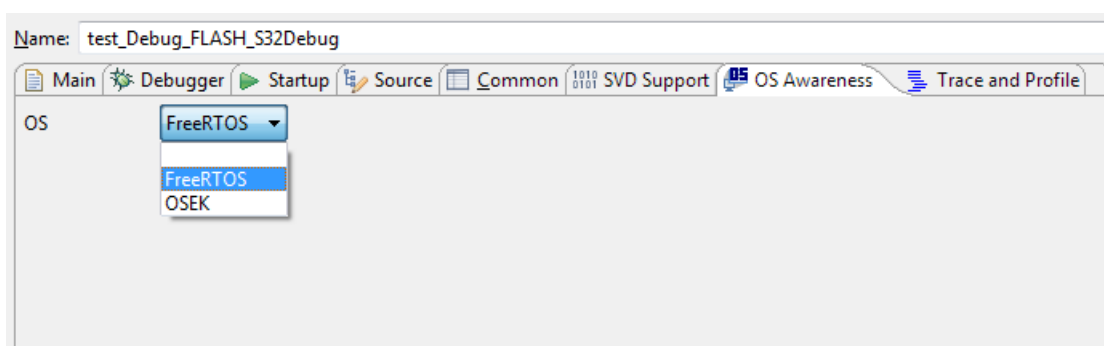


Table 11: Debug Configurations - SVD Support tab

Setting	Description
Clear content of Watch Registers view	<p>Enable if you want an automatic clearing of all info about registers used in the Watch registers view in previous debug session with the same SVD files. If you want system to ask confirmation for any clear, enable the Ask before clear checkbox. Default: Disabled.</p> <p>Note: If the option is enabled, registers info will be cleared for all configurations that use the same SVD files.</p>
SVD source	<p>Specify the SVD file to be used. Default: Use default SVD file.</p> <p>If required, uncheck the Use default SVD file checkbox and select one of these options:</p> <ul style="list-style-type: none"> • Specify the path manually: Type in a path to the location of the SVD file, • Select Device: Specify the path via specifying the device, • Browse Workspace: Specify the path browsing the workspace, • Browse File System: Specify the path browsing the file system, • Variables: Specify the path via setting the variable. <p>The new path to the SVD file appears in the SVD source field.</p>
Support Arm Core registers	<p>Arm SVD source to be used. Default: Enabled (if available).</p>

OS Awareness tab

The **OS Awareness** tab allows you to inform the debugger about the operating system (OS) running on the target hardware: FreeRTOS or OSEK. The debugger provides additional functionality specific to the selected operating system.



After the operating system is selected, the Debug perspective starts to display the **OS Resources** view.

Trace and Profile tab

The **Trace and Profile** tab allows you to configure collection of code trace information. This tab appears in debug configurations intended for debugging with S32 Debug Probe.

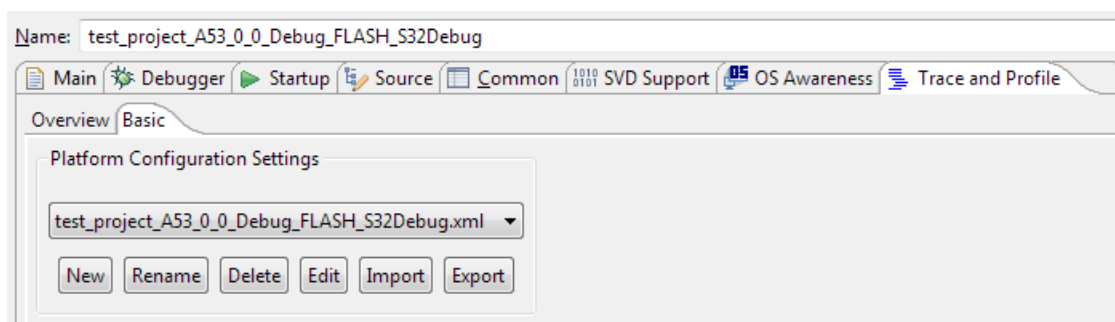


Table 12: Debug Configurations - Trace and Profile settings

Setting	Description
Overview	Click this page to view the flow chart for collecting code trace. The information on this page is read-only.
Basic	<p>Click this page and expand the list to select a configuration file for trace collection. You can click New and create a new file, or you can select an existing configuration file.</p> <p>To learn more, go to Help > Help Contents > S32DS Software Analysis Documentation. The PDF version of this documentation is available in folder <code><S32DS install path>/S32DS/help/pdf</code>.</p>

Running a launch configuration

To start debugging an application in S32 Design Studio for S32 Platform, run the appropriate launch configuration belonging to the application project.

To run a launch configuration:

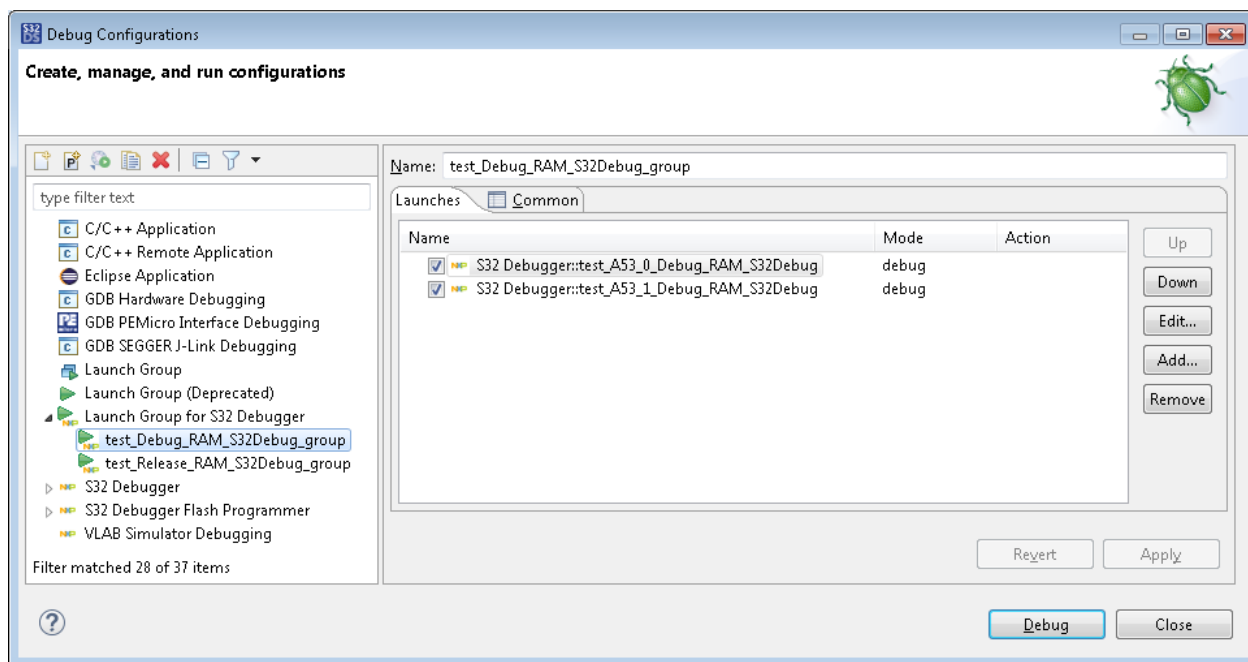
1. Right-click the project in the **Project Explorer** and click **Debug As > Debug Configurations** on the context menu.
2. In the left pane of the **Debug Configurations** dialog box, expand the debugging interface to be used in the debug session.

3. Below the selected debugging interface, click the launch configuration intended for your project and serving your task. The selected configuration appears in the right pane.
4. Update the debug configuration settings as required. For details, refer to the [Editing a launch configuration](#) section.
5. Click **Apply** to save the recent updates to the configuration.
6. Click **Debug** to start a debug session.

Using launch groups

A launch group is a collection of debug configurations. To debug software on a multi-core target, you need to launch several debug configurations, each starting a debug session for one core. Launch groups automate this task by launching all included debug configurations one after another with the predefined conditions.

When you create an application project for a multi-core target in S32 Design Studio for S32 Platform, the required launch groups are generated automatically and are available in the **Debug Configurations** dialog box:



The name of a generated launch group has the following format: “<project_name>_<build configuration>_<debugger>_group”.

A click on a launch group in the left pane displays its structure in the right pane:



- The top configuration is launched first. This configuration initializes the boot core and requires a post-launch delay. This configuration can load the code to flash memory (“Debug_FLASH”) or to RAM (“Debug_RAM”).
- The next configurations are ranged in the order of launching and can be run one after another without a delay, starting debug sessions for the cores in the order of indexing. These configurations load the code to RAM.

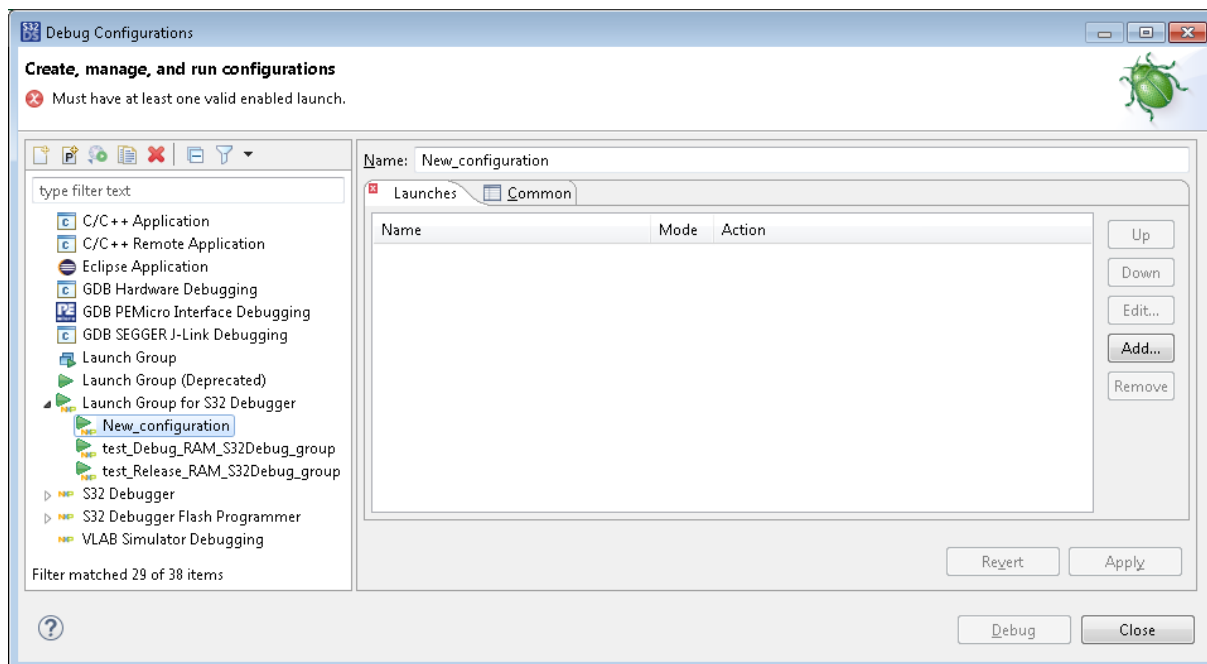
To learn more about using launch groups, refer to the following topics:

- [Creating a launch group](#)
- [Running a launch group](#)

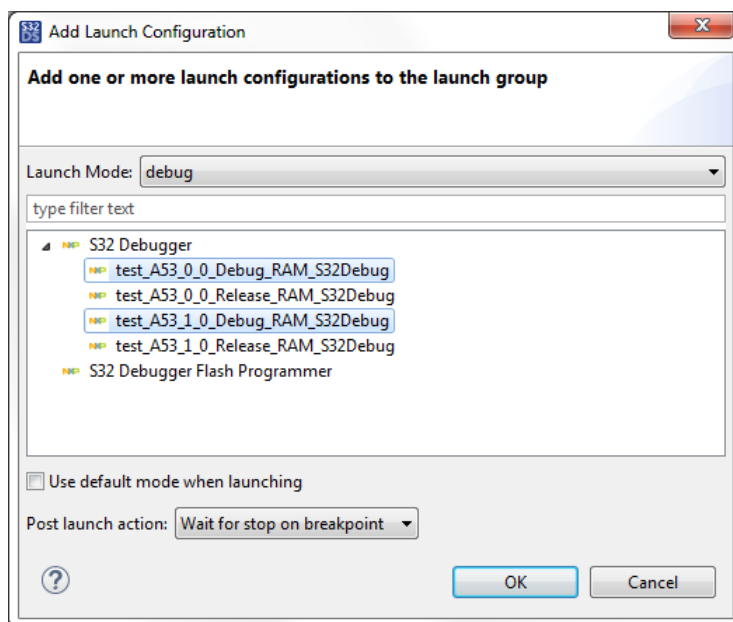
Creating a launch group

To create a launch group:

1. Click the  (*Debug As*) button on the toolbar, then click **Debug Configurations** from the drop-down menu.
2. In the **Debug Configurations** dialog box, click **Launch Group** or **Launch Group for S32 Debugger** in the left pane. Click the  (*New launch configuration*) toolbar button in the left pane. A new launch group configuration appears in the left pane:



3. Click the new launch group in the left pane. Specify the name of the new launch group in the right pane and click **Apply**.
4. In the right pane, go to the **Launches** tab and click **Add**.
5. In the **Add Launch Configuration** dialog box, click the debug configuration to be added to the launch group. To add several configurations, click them, keeping the **Ctrl** key pressed.



6. Expand the **Post launch action** menu and specify the action to be taken before the next configuration is launched:

- **None:** The next configuration will be launched after the current one without a delay.
 - **Wait until terminated:** The next configuration will be launched right after termination of the debug session spawned by the current configuration.
 - **Wait for stop on breakpoint:** The next configuration will be launched right after the breakpoint is hit, for example, at the end of initialization section. This option is available for **Launch Group for S32 Debugger** only.
 - **Delay:** The next configuration will be launched after the current one with the delay specified in seconds.
7. Click **OK**. The launch group now includes the selected debug configurations. Click **Apply**.
 8. To edit any entry in the launch group, click that entry and click **Edit**. Make your updates in the **Edit Launch Configuration** dialog box and click **OK**.
 9. Click **Apply** in the **Debug Configurations** dialog box when your launch group is finished.

Running a launch group

To run a launch group, open it in the **Debug Configurations** dialog box and click the **Debug** button.

- S32 Design Studio for S32 Platform loads the **Debug** perspective.
- The debugger initiates the debug session for the boot core and sends the status of the operation to the **Console** view. If the operation is successful, the started debug session appears in the **Debug** view.
- After the post-launch action, the debugger runs the remaining debug configurations in the launch group and initiates the debug sessions for the remaining cores. If started successfully, the secondary debug sessions appear in the **Debug** view.

To continue debugging, refer to [Debugging on multiple cores](#).

Debugging on a bare-metal target

In S32 Design Studio for S32 Platform, you can debug embedded applications on the chip. The evaluation board with the built-in MCU (target) is connected to the developer's workstation over USB or Ethernet using a JTAG compatible hardware debug probe. The supported probes are listed in topic [Selecting a hardware debug probe](#).

Debugging on a bare-metal target can be performed from RAM and from flash. When you create an application project in S32 Design Studio for S32 Platform, the project creation wizard generates configurations for debugging from both these types of on-chip memory. To choose the right memory type for debugging, take into account memory size and the application size. To start a debug session from the preferred memory type, just launch the appropriate debug configuration as described in the following topics:

- [Debugging with S32 Debug Probe from RAM](#)
- [Debugging with S32 Debug Probe from flash for S32V23x targets](#)
- [Debugging with S32 Debug Probe from flash for other targets](#)
- [Debugging with a PEMicro probe](#)
- [Debugging with a Lauterbach probe](#)

The on-chip debugging techniques available to you are those supported by the hardware debugger. In the common case, you can use hardware breakpoints and do step-by-step execution in the code and in the disassembler instructions.

In addition, you can view and edit the contents of the registers and memory sections. These techniques are described in the following topics:

- [Viewing Registers](#)
- [Viewing memory](#)

When debugging a program from flash, you have an option to manage flash memory from the command line. To learn more, refer to section [Managing flash memory](#).

When running a program on the target connected with S32 Debug Probe, you can collect code trace information. Trace collection can be configured and the collected results analyzed in S32 Design Studio for S32 Platform as described in **Help > Help Contents > S32DS Software Analysis Documentation**.

Selecting a hardware debug probe

To debug an application on a bare-metal target, you need to connect the evaluation board with the target to a computer running S32 Design Studio for S32 Platform. For this purpose, you can use one of the following JTAG compliant hardware debug interfaces:

Table 13: Supported hardware debug probes

Manufacturer	Hardware debug interface (probe)	Details
NXP	<ul style="list-style-type: none"> S32 Debug Probe 	www.nxp.com
PEMicro	<ul style="list-style-type: none"> USB Multilink Universal USB Multilink Universal FX OSBDM/OSJTAG Cyclone TraceLink OpenSDA 	www.pemicro.com
Lauterbach	<ul style="list-style-type: none"> PowerDebug USB 3 PowerDebug Pro µTrace for Cortex-M 	www.lauterbach.com

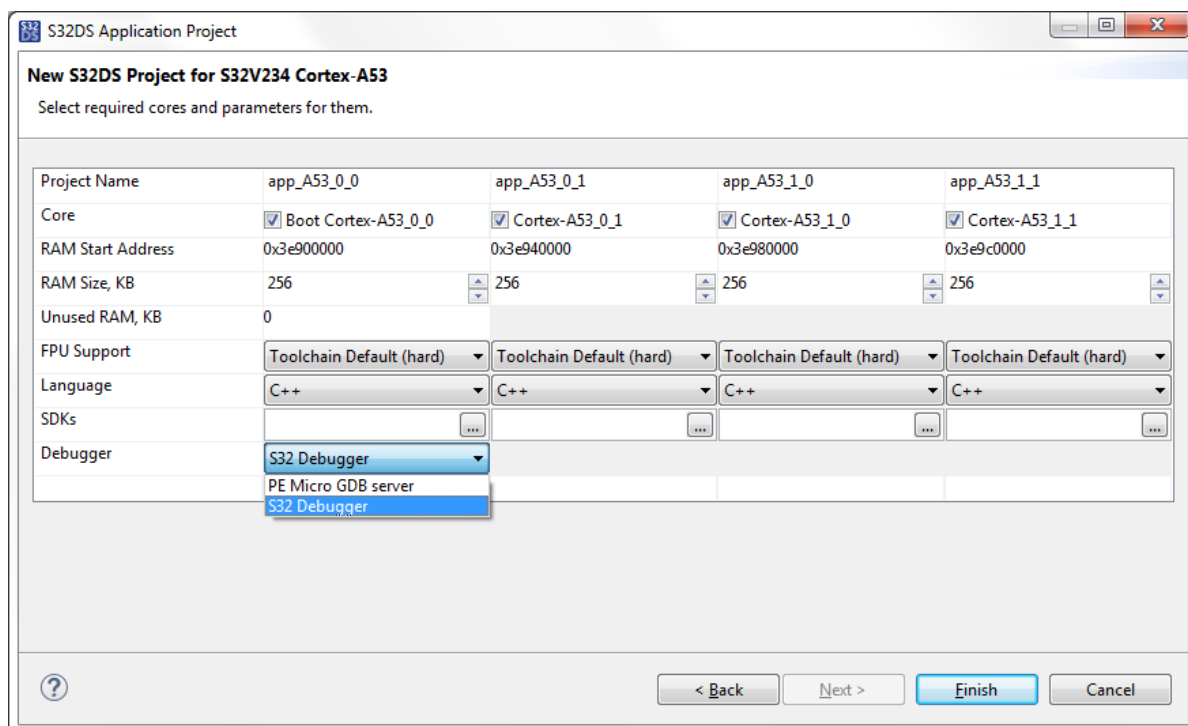
Note: The scope of supported MCU families and cores differs for each model and shall be checked on the manufacturer's website.

Debugging with S32 Debug Probe from RAM

This topic describes how to debug a bare-metal application on the board connected to the computer with the S32 Debug Probe. The application is loaded by S32 Debugger to RAM memory of the target.

To debug an application with the S32 Debug Probe from RAM:

1. Create an application project in the wizard and specify “S32 Debugger” as the debugger.



2. Connect the board to your computer. To learn the details about S32 Debug Probe, refer to the *S32 Debug Probe User Guide*. The PDF version of this document is located in the `/S32DS/tools/S32Debugger/Debugger/docs/` folder.
3. Build the project.
4. Open the **Debug Configurations** dialog and go to the **S32 Debugger** group of configurations. Click the debug configuration for debugging from RAM generated for your project.
5. On the **Main** tab, verify the following settings:
 - **Project:** The path of the application project.
 - **Application:** The path of the executable (ELF) built from the application project.
 - **Build Configuration:** The build configuration for debugging from RAM. If you activate the **Enable auto build** option, this build configuration will be used to rebuild the application before each debug session.
6. On the **Debugger** tab, specify the following settings:

Name: test_project_A53_0_0_Debug_RAM_S32Debug

Main Debugger Startup Source Common SVD Support OS Awareness Trace and Profile

Hardware
 Device: S32V234 Core: A53_0_0
 Initialization script: \${S32DS_INITIALIZATION_SCRIPTS_DIR}/s32v234/s32v234_generic_bareboard.py
 Initial core

Debug Probe Connection
 Interface: S32 Debug Probe - Ethernet
 USB device:
 Hostname or IP: localhost

Target Communication Speed
 JTAG Speed (KHz): 16000 Timeout: 30 s
 Delay after reset: 3000 ms

GDB Server
 Launch server
 Server port number: 45000
 Enable log

GDB Client
 Executable: \${S32DS_GDB_ARM64_PY}
 Commands:
 Force thread list update on suspend

Semihosting
 Enable semihosting
 Port: 45001

Secure debugging
 Enable secure debugging
 Debugging type:

Table 14: Debugger tab: Settings for debugging with S32 Debug Probe

Setting	Description
Hardware	<p>Specify the settings related to the target:</p> <ul style="list-style-type: none"> Select device and core: Click this button and pick the target processor and the core from the list. The data appears in the Device and Core fields. Initialization script: The path of the generic initialization script generated automatically. <p>To specify the script particular to your evaluation board, go to the following folder:</p> <pre><S32 Design Studio for S32 Platform installation folder>/S32DS/tools/S32Debugger/Debugger/scripts/s32xxxx</pre> <p>The last folder in the path is named as your target processor, for instance, /s32v234. Find the information about the available script files in the readme file located in this folder.</p>

Setting	Description
	<ul style="list-style-type: none"> • Initial core: Enable this option if the debug configuration will start the first debug session in a launch group or the only debug session in single-core debugging. When enabled, this option indicates that the evaluation board needs to be initialized. <p>In multi-core debugging, make sure to disable this option in all debug configurations that start the second and all later debug sessions in a launch group.</p> <p>Note: When disabled, the Initial core option makes the probe connection settings, JTAG settings, and GDB server settings (below) unavailable.</p>
Debug Probe Connection	<p>Configure the connection between the board and the computer.</p> <ul style="list-style-type: none"> • Interface: Specify the connection interface. • USB device: For the USB connection, specify the COM port to which the board is connected. <p>If the board is not connected with the USB cable, connect it and wait for the probe's TX/RX indicator to get green. Then click Refresh for the connected COM port to appear on the Port menu.</p> <p>If you have a problem with the USB connection:</p> <ul style="list-style-type: none"> • On Windows, install the S32 Debug Probe driver manually. For details, refer to readme.txt in S32DS/tools/S32Debugger/Debugger/drivers/usb/. • On Linux, install the <i>udevadm</i> utility, this tool detects the used port. • Hostname or IP: For the Ethernet connection, specify the host name or IP address of the probe network adapter. <p>Note: To learn the host name of the probe, refer to the documentation provided with the delivery kit. For the static IP address assigned to the probe, consult your network administrator.</p> <ul style="list-style-type: none"> • Test connection: click the button to check the connection.
JTAG Communication Speed	<p>Specify the JTAG communication settings.</p> <ul style="list-style-type: none"> • JTAG Speed (KHz): Specify the JTAG speed. • Timeout: Specify the JTAG timeout. • Delay after reset: Enable this option to perform software reset on the device at the beginning of the debug session. Specify the board initialization delay (in milliseconds).
GDB server	<p>Specify the GDB server settings.</p> <ul style="list-style-type: none"> • Launch server: This option launches the GDB server. Keep it enabled unless the GDB server is started from the command line. • Server port number: Specify the GDB server port. When debugging multiple cores of a single processor, specify the same port in all debug configurations. Default: 45000. • Enable log: Enable this option to log the GDB server output. This option is disabled by default. <p>Note: Logging may greatly reduce the speed of the debugging process.</p>

Setting	Description
GDB Client	<p>Specify the GDB client settings:</p> <ul style="list-style-type: none"> • Executable: The path of the GDB client. This path is generated automatically when you click the Select device and core button and select the target device and the core. • Commands: If required, specify commands to be executed after the GDB client is started. These commands are executed next to the commands specified on the Startup tab. • Force thread list update on suspend: Enable this option for the thread list to be updated forcibly if the debug session is suspended.
Semihosting	<p>Configure semihosting, that is, the ability of the debug session to send output to the Console view.</p> <ul style="list-style-type: none"> • Enable semihosting: Enable semihosting. • Port: The port for semihosting. Read-only. The port number is defined as (GDB server port +1). Default: 45001. <p>Note: If semihosting is enabled, the semihosting console needs to be opened in the Console view once the debug session is started.</p>
Secure debugging	<p>Configure settings for secured chips.</p> <ul style="list-style-type: none"> • Enable secure debugging: Enable this option to start a debug session on locked chip. • Debugging type: Select the authentication method. Options: Password, Challenge & Response. • Clear data stored: Click the button to delete any data from secure storage. <p>Note: The availability and support of the option depends on the target device.</p> <p>Note: If the chip is unlocked, enabling secure debugging is meaningless.</p>

7. On the **Startup** tab, specify the place in code where the first breakpoint will be set (*main* by default). Leave the remaining settings with their default values.
8. Click **Apply**, then click **Debug**. The debug session is started and stopped at the first breakpoint.
9. Debug the application as usual.

Debugging with S32 Debug Probe from flash for S32V23x targets

Note: This section applies only to S32V23x targets.

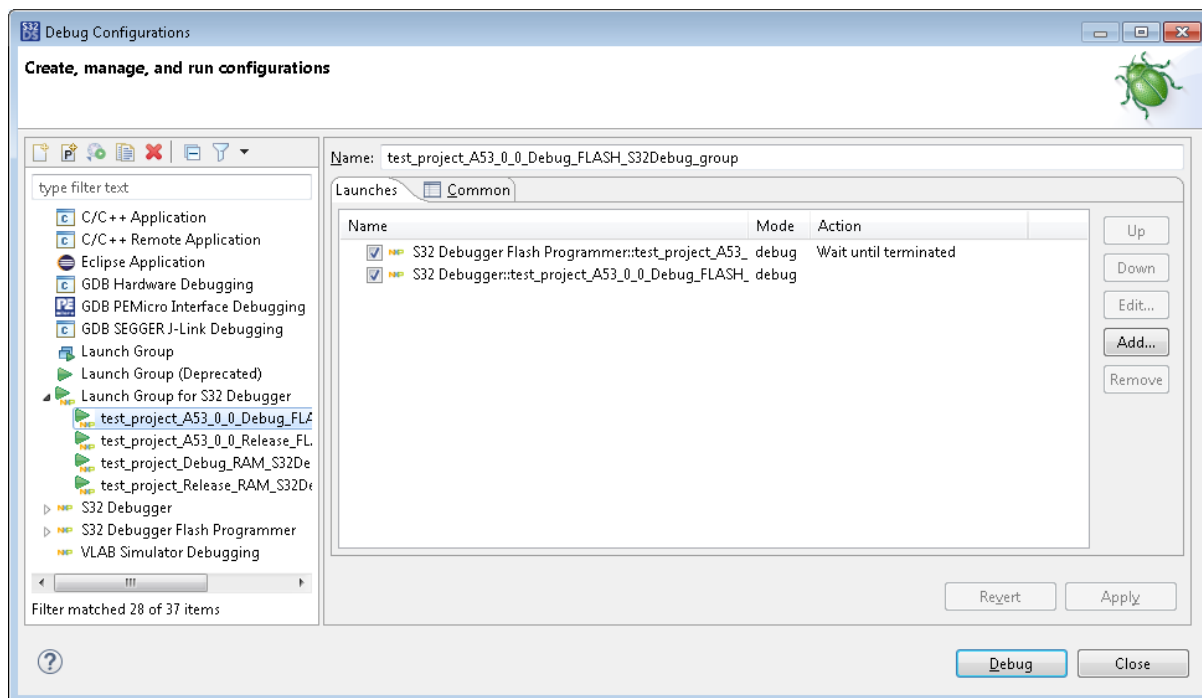
This topic describes how to debug a bare-metal application on the board connected to the computer with the S32 Debug Probe. The application is loaded by S32 Flash Programmer to flash memory external to the target. The application is launched by the S32 Debugger.

To debug an application with the S32 Debug Probe from flash:

1. Create an application project and specify “S32 Debugger” as the debugger.
2. Build the project.

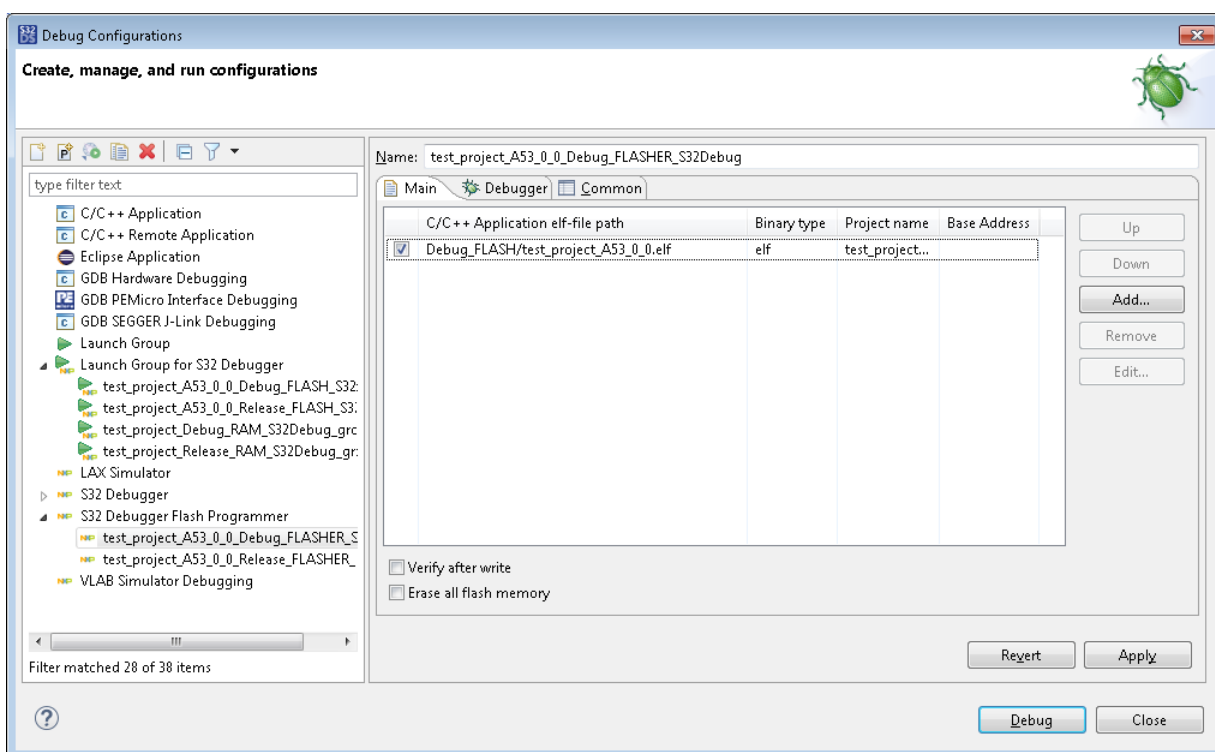
If you have selected multiple cores when creating your project, build the project for each selected core.
3. Right-click the project in the **Project Explorer** and click **Debug As > Debug Configurations** on the context menu.
4. In the **Debug Configurations** dialog box, go to the **Launch Group** section in the left pane.

The project creation wizard has generated several launch groups for your application. In the left pane, click the launch group intended for debugging from flash:



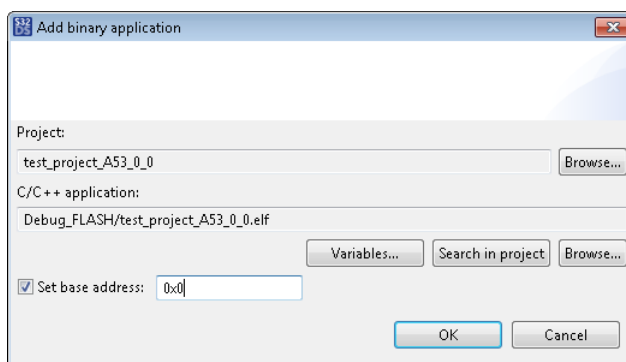
The right pane displays the debug configurations in the order of execution. The top configuration is intended for loading the executable code to flash memory of the target. The “Wait until terminated” action indicates that the next debug configuration will be run only after the flash loading session has finished. To learn more about using launch groups, refer to [Creating a launch group](#).

5. Expand the **S32 Debugger Flash Programmer** interface in the left pane and click the debug configuration that is located on top in the launch group.



- On the **Main** tab you can add, remove, edit and change order of binary elf-files.

To add new binary file press **Add** button. **Add binary application** dialog will appear. Specify the details and press **OK**.



- Enable the **Erase all flash memory** option to apply the respective action before writing data to flash.

Note: In order to write a new flash image or debug a new application the board should be placed in serial boot mode.

- Go to the **Debugger** tab and make sure that the following settings are defined properly:

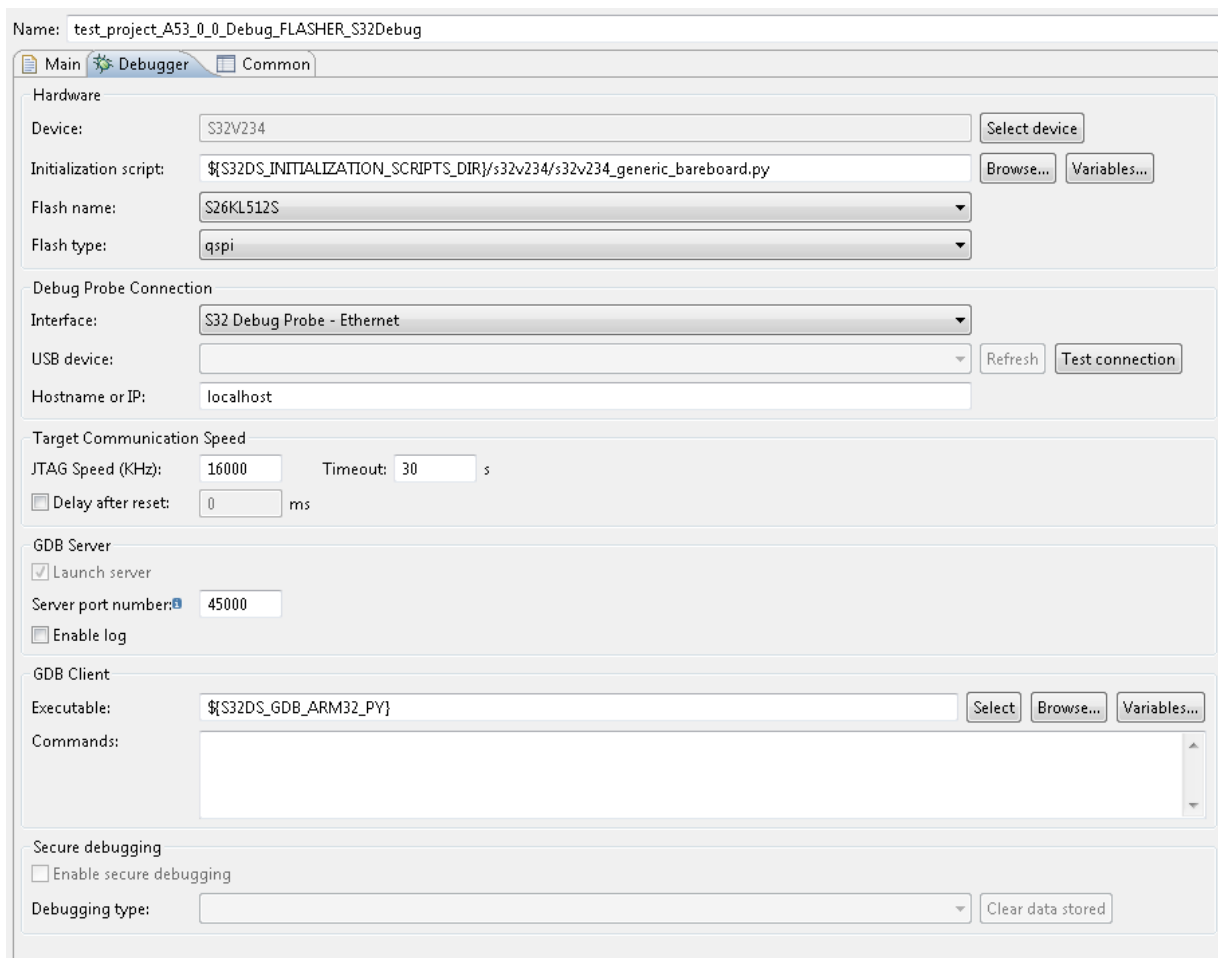


Table 15: Debugger tab: Settings for debugging with S32 Debug Probe

Setting	Description
Hardware	<p>Specify the settings related to the target:</p> <ul style="list-style-type: none"> • Device: This field is populated automatically from the project settings. If correction is needed, use the Select device button to choose the right device. • Initialization script: The initialization script is populated automatically based on the device information. All configurations in the launch group use the same script file. To specify the script particular to your evaluation board, go to the following folder: <S32 Design Studio for S32 Platform installation folder>/S32DS/tools/S32Debugger/Debugger/scripts/s32xxxx The last folder in the path is named as your target processor, for instance, /s32v234. Find the information about the available script files in the README file located in this folder. • Flash name and Flash type: Expand the menus and select the flash device and the type.

Setting	Description
Debug Probe Connection	Configure the connection between the board and the computer: <ul style="list-style-type: none"> • Interface: Specify the connection interface. • USB device: For the USB connection, specify the COM port to which the board is connected. If the board is not connected with the USB cable, connect it and wait for the probe's TX/RX indicator to get green. Then click Refresh for the connected COM port to appear on the Port menu. <p>If you have a problem with the USB connection:</p> <ul style="list-style-type: none"> • On Windows, install the S32 Debug Probe driver manually. For details, refer to <code>readme.txt</code> in <code>S32DS/tools/S32Debugger/Debugger/drivers/usb/</code>. • On Linux, install the <code>udevadm</code> utility, this tool detects the used port. • Hostname or IP: For the Ethernet connection, specify the host name or IP address of the probe network adapter. <p>Note: To learn the host name of the probe, refer to the documentation provided with the delivery kit. For the static IP address assigned to the probe, consult your network administrator.</p> <ul style="list-style-type: none"> • Test connection: click the button to check the connection.
Target Communication Speed	Specify the JTAG communication settings: <ul style="list-style-type: none"> • JTAG Speed (KHz): Specify the JTAG speed. • Timeout: Specify the JTAG timeout. • Delay after reset: Enable this option to perform software reset on the device at the beginning of the debug session. Specify the board initialization delay (in milliseconds).
GDB server	Specify the GDB server settings: <ul style="list-style-type: none"> • Launch server: This option launches the GDB server. Always enabled. • Server port number: Specify the GDB server port. When debugging multiple cores of a single processor, specify the same port in all debug configurations. Default: 45000. • Enable log: Enable this option to log the GDB server output. This option is disabled by default. <p>Note: Logging may greatly reduce the speed of the debugging process.</p>
GDB Client	Specify the GDB client settings: <ul style="list-style-type: none"> • Executable: The path of the GDB client. This path is generated automatically. And can be changed automatically when you click the Select device button and select the target device. • Commands: If required, specify commands to be executed after the GDB client is started. These commands are executed next to the commands specified on the Startup tab.
Secure debugging	Configure settings for secured chips: <ul style="list-style-type: none"> • Enable secure debugging: Enable this option to start a debug session on locked chip.

Setting	Description
	<ul style="list-style-type: none"> • Debugging type: Select the authentication method. Options: Password, Challenge & Response. • Clear data stored: Click the button to delete any data from secure storage. <p>Note: The availability and support of the option depends on the target device.</p> <p>Note: If the chip is unlocked, enabling secure debugging is meaningless.</p>

9. Click **Apply** to save your updates to the debug configuration.
10. Expand the **S32 Debugger** interface in the left pane and click the debug configuration that follows the top one in the launch group. Specify the settings as described in the above step.

The following settings are required if the debug configuration is the first one to be executed after the flash loading session:

- **Initial core:** Keep this option enabled for the evaluation board to be initialized.
 - **Launch server:** Keep this option enabled for the GDB server to be started. Remove this flag only if you are going to run the GDB server from the command line.
11. If the launch group includes more debug configurations for other cores, open them from the left pane under **S32 Debugger** as described above. Make sure that all these configurations have the **Initial core** option not selected.
 12. Go back to the launch group and click **Debug**.
 13. When the debug session is started, the execution stops at the first software breakpoint (typically, at *main*) that is specified on the **Startup** tab in the **Set breakpoint at** field. Step over and continue debugging your program on the target as you always do.

Debugging with S32 Debug Probe from flash for all other targets

This topic describes how to debug a bare-metal application on the board connected to the computer with the S32 Debug Probe. The application is loaded by S32 Flash Programmer to flash memory external to the target. The application is loaded from flash memory by the target to RAM memory of the target and launched. The S32 Debugger connects to the running application by Attach method.

To debug an application with the S32 Debug Probe from flash:

1. Create an application project in the wizard and specify “S32 Debugger” as the debugger.
2. Generate S-Record/Intel HEX/ Binary file selecting the **Raw Binary** option.
 - Right click on the Project name in **Project Explorer**.
 - Select **Properties > C/C++ Build > Settings > Cross Settings**.
 - Check the **Create flash image** checkbox.
 - Click **Apply and Close** button.
 - Reopen project settings and go to the **Standard S32DS Create Flash Image > General**.
 - Select **Raw binary** option for **Output file format**.
 - Click **Apply and Close** button.
3. Connect the board to your computer. To learn the details about S32 Debug Probe, refer to the *S32 Debug Probe User Guide*. The PDF version of this document is located in the `/S32DS/tools/S32Debugger/Debugger/docs/` folder.
4. Build the project generating the binary executable.

This will be your application binary input to the IVT Tool.
5. Generate the BLOB image which can be programmed to flash memory device and loaded to the RAM by the BootROM using the IVT Tool.

Note: S32 Flash Programmer supports only IVT image binaries. For help on obtaining IVT image for your project refer to *IVT Tool* section in **Help > Help Contents > S32 Configuration Tool Getting Started**.

The resulting BLOB image file is what can be flashed to the device.

6. Open the **Debug Configurations** dialog and double-click the **S32 Debugger Flash Programmer**.
7. Specify the new configuration name.
8. On the **Main** tab press **Add** button to add new binary file. **Add binary application** dialog will appear.
 - a) Click **Browse** button.
 - b) Select the project from the workspace where the application binary is located.
 - c) Click **OK** button.

By default, the ELF file is found.

- d) Click **Search in project** button
 - e) Select the binary file (IVT image obtained at step 5).
 - f) Click **OK** button.
 - g) Enter the base address to the **Set base address** field.

Note: Typically, this could be 0, but you may have other requirements
 - h) Click **OK** button.
9. Check the **Erase all flash memory** checkbox if needed.

Note: Just the memory required by the new image needs to be cleared.

10. On the **Debugger** tab specify the following settings:

Table 16: Debugger tab: Settings for debugging with S32 Debug Probe from flash

Setting	Description
Hardware	<p>Specify the settings related to the target:</p> <ul style="list-style-type: none"> • Device: This field is populated automatically from the project settings. If correction is needed, use the Select device button to choose the right device. • Initialization script: The path of the generic initialization script is generated automatically. <p>To specify the script particular to your evaluation board, go to the following folder:</p> <pre><S32 Design Studio for S32 Platform installation folder>/S32DS/tools/S32Debugger/Debugger/scripts/s32xxxx</pre> <p>The last folder in the path is named as your target processor. Find the information about the available script files in the README file located in the folder.</p> <ul style="list-style-type: none"> • Flash name and Flash type: The flash device and the type will automatically be set.
Debug Probe Connection	<p>Configure the connection between the board and the computer to match your setup.</p> <ul style="list-style-type: none"> • Interface: Specify the connection interface. • USB device: For the USB connection, specify the COM port to which the board is connected. <p>If the board is not connected with the USB cable, connect it and wait for the probe's TX/RX indicator to get green. Then click Refresh for the connected COM port to appear on the Port menu.</p> <p>If you have a problem with the USB connection:</p>

Setting	Description
	<ul style="list-style-type: none"> On Windows, install the S32 Debug Probe driver manually. For details, refer to readme.txt in S32DS/tools/S32Debugger/Debugger/drivers/usb/. On Linux, install the <i>udevadm</i> utility, this tool detects the used port. <p>Hostname or IP: For the Ethernet connection, specify the host name or IP address of the probe network adapter.</p> <p>Note: To learn the host name of the probe, refer to the documentation provided with the delivery kit. For the static IP address assigned to the probe, consult your network administrator.</p> <ul style="list-style-type: none"> Test connection: click the button to check the connection.
Target Communication Speed	<p>Specify the JTAG communication settings.</p> <ul style="list-style-type: none"> JTAG Speed (KHz): Specify the JTAG speed. Timeout: Specify the JTAG timeout. Delay after reset: Enable this option to perform software reset on the device at the beginning of the debug session. Specify the board initialization delay (in milliseconds).
GDB server	<p>Specify the GDB server settings.</p> <ul style="list-style-type: none"> Launch server: This option launches the GDB server. Keep it enabled unless the GDB server is started from the command line. Server port number: Specify the GDB server port. When debugging multiple cores of a single processor, specify the same port in all debug configurations. Default: 45000. Enable log: Enable this option to log the GDB server output. This option is disabled by default. <p>Note: Logging may greatly reduce the speed of the debugging process.</p>
GDB Client	<p>Specify the GDB client settings:</p> <ul style="list-style-type: none"> Executable: The path of the GDB client. This path is generated automatically when you click the Select device button and select the target device. Commands: If required, specify commands to be executed after the GDB client is started. These commands are executed next to the commands specified on the Startup tab.
Secure debugging	<p>Configure settings for secured chips.</p> <ul style="list-style-type: none"> Enable secure debugging: Enable this option to start a debug session on locked chip. Debugging type: Select the authentication method. Options: Password, Challenge & Response. Clear data stored: Click the button to delete any data from secure storage. <p>Note: The availability and support of the option depends on the target device.</p> <p>Note: If the chip is unlocked, enabling secure debugging is meaningless.</p>

11. Click **Apply** to save your updates to the debug configuration.

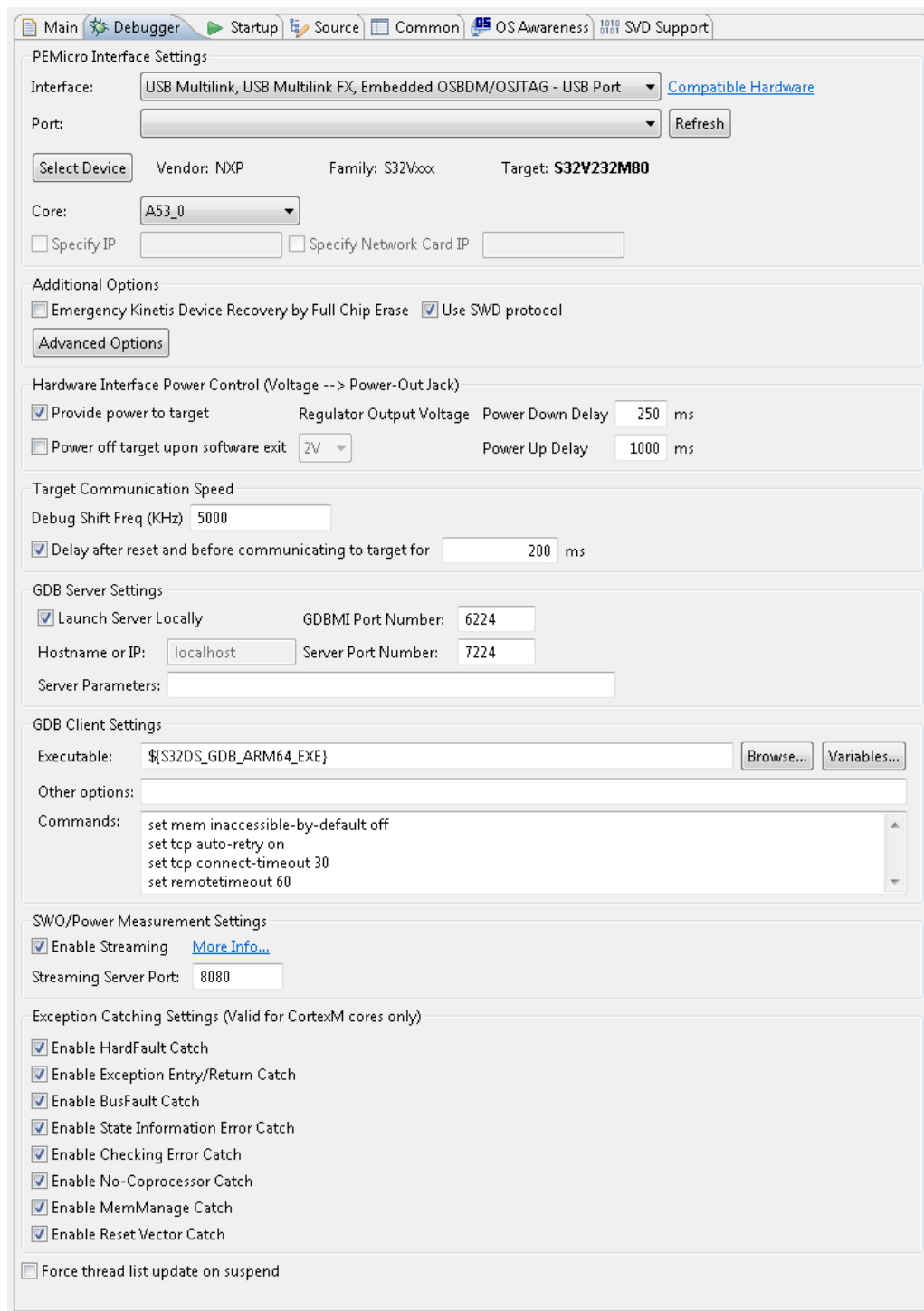
12. Expand the **S32 Debugger** interface in the left pane and click the debug configuration corresponding to your application project since it will be subsequently started by the BootROM.
13. On the **Debugger** tab specify the `<device>_attach.py` script in the **Initialization script** field.
Note: For all other **Debugger** tab setting refer to the [Debugging with S32 Debug Probe from RAM](#) topic.
14. On the **Startup** tab specify the following settings:
 - Uncheck the **Load image** checkbox.
 - Check the **Set program counter at** checkbox and enter the value “Reset_Handler”.
15. Click **Apply** to save your updates to the debug configuration.
16. Click **Debug**. The debug session is started.
17. When completed, the terminated thread will be shown in the **Debug perspective**.
18. Continue debugging.

Debugging with a PEMicro probe

Debugging on a bare-metal target with a PEMicro probe requires the PEMicro GDB server (Eclipse plug-in) to be installed on S32 Design Studio for S32 Platform. Find the details in topic [Installing plug-ins](#).

To debug a bare-metal application on the board connected to the computer with a PEMicro probe:

1. Create an application project in the wizard and specify “GDB PEMicro Debugging Interface” as the debugger.
2. Build the application.
3. Open the **Debug Configurations** dialog and go to the **GDB PEMicro Interface Debugging** group of configurations. Click the debug configuration for debugging from RAM or from flash generated for your project.
4. On the **Debugger** tab, specify the following mandatory settings:



- **Interface:** Select the PEMicro debug interface connected to the target device. Click **Compatible Hardware** to open the PEMicro Web page and view the list of targets supported by the selected debug interface.
- **Port:** If the debug interface uses a USB connection, select the COM port labeled with the device name from the **Port** menu. Click **Refresh** to update the list of connected ports.
- **Target:** Click **Select Device** and select the target device. The vendor, family, and type of the device appear in the respective fields.
- **Core:** Expand the **Core** list and select the target core.

- **Executable:** Select or enter the path to the GDB client.

Leave the remaining settings with their default values.

Note: To set up your configuration for a special debugging scenario, consult the P&E GDB Server Plug-In debug configuration user guide. This document is available in the <S32 Design Studio for S32 Platform installation path>/S32DS/help/pdf/ folder.

5. Click **Apply**, then click **Debug**. The debug session is started and stopped at the first breakpoint (typically, at *main*) that is specified on the **Startup** tab in the **Set breakpoint at** field.
6. To use real-time printf with Instrumentation Trace Macrocell (supported for particular devices), click **Window** > **Show View** > **Other...** > **PEmicro** > **SWO Printf**. In the SWO Printf console, click the **Start Trace** (🎧) button.
7. Click **Resume** and debug the application as you always do.

Note: When you use the GDB console commands to control the program, views are not updated instantly. Use the `step` command to see the refreshed views.

Debugging with a Lauterbach probe

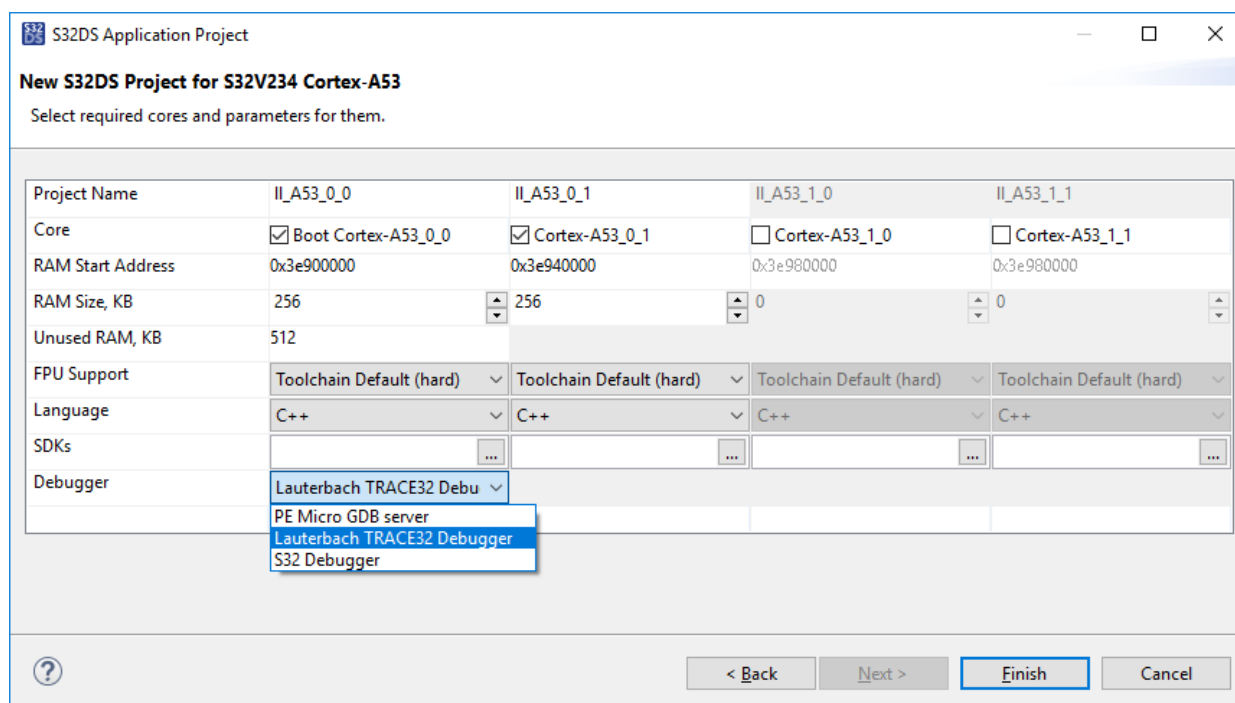
Debugging on a bare-metal target with a Lauterbach probe requires the Lauterbach software to be installed on S32 Design Studio for S32 Platform. Click **Help** > **Install New Software...** to install the Lauterbach TRACE32 Eclipse plug-in. For details, refer to [Installing plug-ins](#).

Note: If you changed the default memory configuration in the linker script, make sure to update the CMM file accordingly. In the project folder, go to **Project Settings** > **Debugger** and open the <device>.cmm file. This file contains the cores initialization, update the boot address line:

```
Data.Set SD:0x4008814C %Long 0x38000000 ; Write boot address to
MC_ME.PRTN0_CORE0_ADDR.R
```

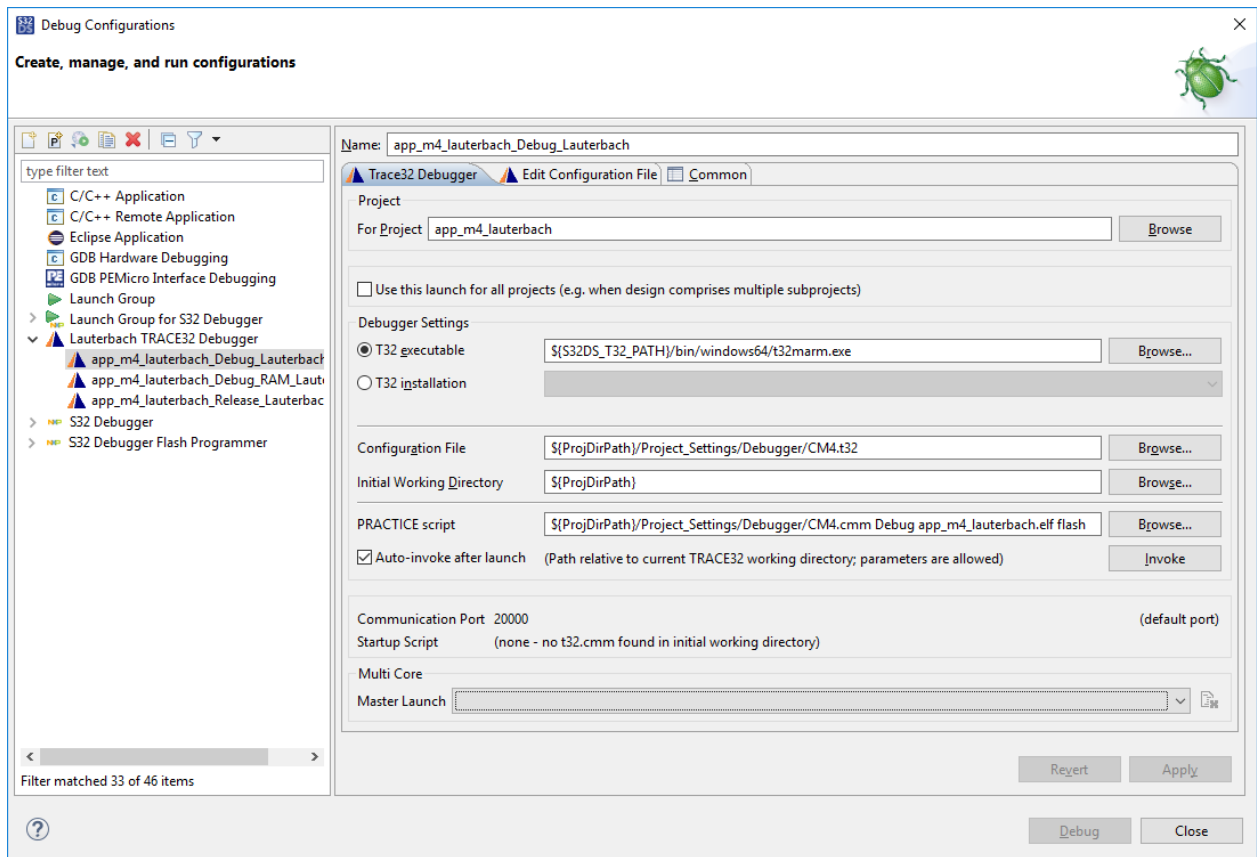
To debug a bare-metal application on the board connected to the computer with a Lauterbach probe:

1. Create an application project in the wizard and specify “Lauterbach TRACE32 Debugger” as the debugger.



2. Build your project.

- Open the **Debug Configurations** dialog and go to the **Lauterbach TRACE32 Debugger** group of configurations. Click the debug configuration for debugging from RAM or from flash generated for your project.



- On the **Trace32 Debugger** tab, specify the connection parameters. The configuration settings are described in www2.lauterbach.com/pdf/int_eclipse.pdf.
- Click **Apply**, then click **Debug**.

When you launch the debug configuration, S32 Design Studio for S32 Platform redirects you to the Lauterbach TRACE32 debugging tool. Debugging in the Lauterbach environment is described in www2.lauterbach.com/pdf/int_eclipse.pdf. When the debug session is terminated, you get back to S32 Design Studio for S32 Platform.

Viewing Registers

There are three types of registers available the S32 Design Studio for S32 Platform:

- processor registers,
- memory mapped registers: peripheral and Arm core (if available for your target),
- Arm system registers (if available for your target).

When in a debug session, use:

- The **Registers view** to display and modify the processor registers values of the connected target. Find the details in the [Viewing processor registers](#) topic.
- The **Peripheral Registers view** to see all the memory mapped registers of the connected target.

Note: You can also use the **EmbSys Registers view** - single-view option to display and modify all peripheral registers listed in the view. Find the details in the [Viewing peripheral registers in EmbSys Registers](#) topic.

Note: S32 Debugger implements different mechanisms for **Peripheral Registers** and **EmbSys Registers** views when accessing SoC memory mapped registers. **Peripheral Registers view** is a preferred option since

S32 Debugger is accessing these registers directly via the system bus, bypassing caches and MMU (if present in the core being debugged).

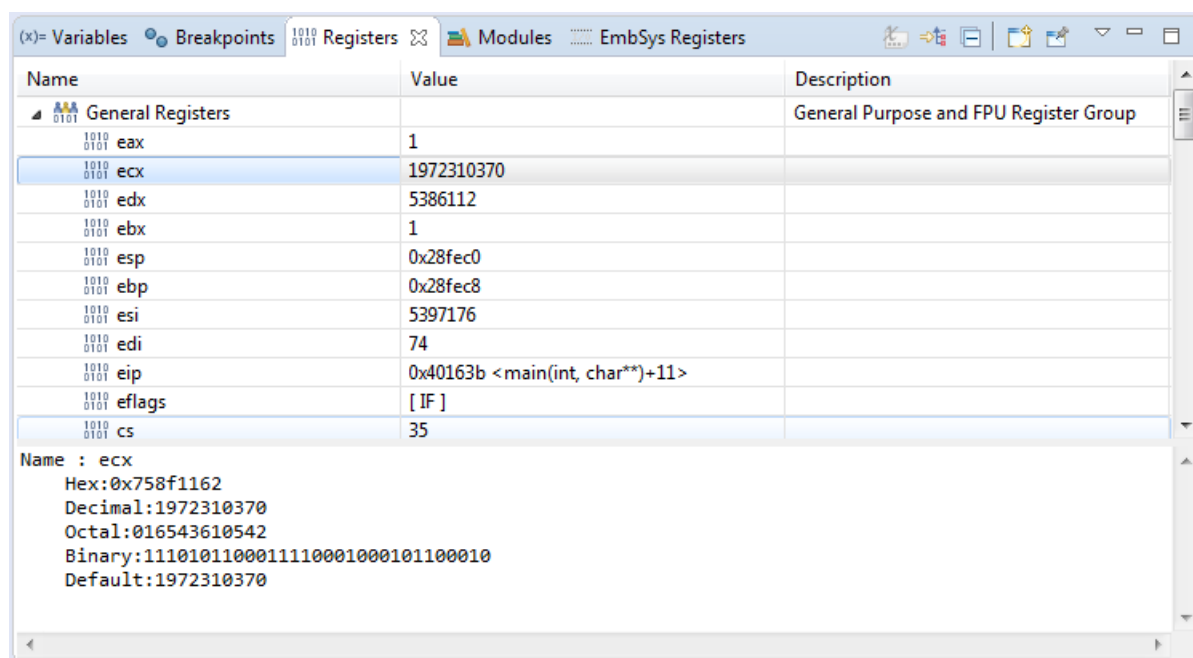
- The [Arm System Registers view](#) to see all the Arm system registers of the connected target.

To access data in the selected memory mapped or Arm system registers use the [Watch registers view](#). Find the details in the following topics:

- [Reading values from registers](#)
- [Setting values to registers](#)
- [Exporting register values](#)
- [Importing register values](#)

Viewing processor registers

When in a debug session, use the [Registers view](#) to display and modify the register values on the connected target. To display the **Registers** view in the **Debug** perspective, click **Window > Show View > Registers** on the menu.



The **Registers** view displays the MCU registers arranged in categories. To display the values of a particular category of registers, expand the respective tree node in the view.

To change the format of the register value, right-click the register in the view and click **Number format** and the preferred format on the context menu. You can choose between the hex, octal, decimal, or binary format.

Reading values from registers

To enable reading of registers when in a debug session use one of the options:

- **Import** a saved set of registers. For detailed information on exporting and importing registers refer to [Exporting register values](#) and [Importing register values](#) topics.
- Manually create a set of registers to interact with:
 1. Open the [Peripheral Registers view](#) or the [Arm System Registers view](#).
 2. Select registers to be watched. A category, a group of registers, a peripheral, a cluster (if set), a register, a field (sequential bits within a register) can be selected. A minimal unit to be watched is a register - if you select a field the whole register will be added.
 3. Press **Enter** or right-click the selection and click **Watch Register(s)** on the context menu or double-click the item.

- The registers selected for reading appear in the [Watch registers view](#).

Peripherals	Hex	Binary	Address	Description
SIUL2_0				
SIUL2_0			0x4009C000	SIUL2
MIDR1	0x1D120011	00011101...	0x4009C004	SIUL2 MCU ID Register #1
Arm_system_registers				
Address_translation_instructions				Address_translation_instructions
PAR_EL1	0x000000003430...	00000000...		Returns the output address (OA) fr...
ATTR (bits 56-63)	0x00	00000000		Memory attributes for the returne...
Reserved (bits 52-55)	0x0	0000		Reserved.
PA_51_48 (bits 48-51)	0x0	0000		Extension to PA[47:12]. See PA[47:...
Reserved_1 (bits 48-51)	0x0	0000		Reserved.
PA_47_12 (bits 12-47)	0x000034303	00000000...		Output address. The output addre...
Reserved_2 (bit 11)	0x1	1		Reserved.
IMPLEMENTATION_DEFINED (bit 10)	0x0	0		IMPLEMENTATION DEFINED.
NS (bit 9)	0x0	0		Non-secure. The NS attribute for a...
SH (bits 7-8)	0x0	00		Shareability attribute, for the retur...
Reserved_3 (bits 1-6)	0x00	000000		Reserved.
F (bit 0)	0x0	0		Indicates whether the instruction p...

The data are read from the registers of the RO (read-only) and RW (read-write) types at each debug action like a step, a resume, a stop, or a breakpoint. When the **Hex** and **Binary** cells are updated with the actual values of the registers, these values are displayed in yellow background.

Note: The **Read on demand** registers provide no values by default and display the “?” character on the initial appearance. To read a **Read on demand** register, select a register(s), right-click and select **Read** on the context menu.

To remove selected registers from the **Watch registers view** press **Delete** or right-click the selection and click **Remove Register(s)** on the context menu.

Setting values to registers

To set a value to RW or WO (write-only) register when in a debug session in the **Watch registers view** use one of the options:

- Click a field's/register's **Hex** cell and start typing. If an enumerated value is set for a field/register you can select a value from the drop-down list with the descriptions shown.

Note: If a field's read/set value is included in the enumerated values, the **Description** cell is replaced with a description of that value from the enumerated values and marked with a symbol.

Peripherals	Hex	Binary	Address	Description
AIPS/MPRA	0x77700000	01110111...	0x40000000	Master Privilege Register A
MPL2 (bit 20)	to user-mode.	1		Accesses from this master are not forced to user-mode.
MTW2 (bit 21)	0	0		Accesses from this master are forced to user-mode. Master is trusted for write accesses.
MTR2 (bit 22)	1	1		Accesses from this master are not forced to user-mode. Master is trusted for read accesses.
MPL1 (bit 24)	0x1	1		Accesses from this master are not forced to user-mode.
MTW1 (bit 25)	0x1	1		This master is trusted for write accesses.
MTR1 (bit 26)	0x1	1		This master is trusted for read accesses.
MPL0 (bit 28)	0x1	1		Accesses from this master are not forced to user-mode.
MTW0 (bit 29)	0x1	1		This master is trusted for write accesses.
MTR0 (bit 30)	0x1	1		This master is trusted for read accesses.
PTE/PTOR	(write only)	(write only)	0x400FF10C	Port Toggle Output Register
PTTO (bits 0-31)	(write only)	(write only)		Port Toggle Output
PTE/PDIR	0x00000000	00000000...	0x400FF110	Port Data Input Register
PDI (bits 0-31)	0x00000000	00000000...		Port Data Input

- Click a field's/register's **Binary** cell, set value and click the **Set** button.


Peripherals	Hex	Binary	Address	Description
1010 0101 EmbeddedRAM0	0x00000000	00000000..	0x14000800	CSE PRAM 0 Register
1010 0101 BYTE_3 (bits 0-7)	0x00	0 0 0 0 . 0 0 0 0	Set	Data byte 3 of Rx/Tx frame.
1010 0101 BYTE_2 (bits 8-15)	0x00	00000000		Data byte 2 of Rx/Tx frame.
1010 0101 BYTE_1 (bits 16-23)	0x00	00000000		Data byte 1 of Rx/Tx frame.
1010 0101 BYTE_0 (bits 24-31)	0x00	00000000		Data byte 0 of Rx/Tx frame.
1010 0101 ACTLR	0x00000000	00000000...	0xE000E008	Auxiliary Control Register,

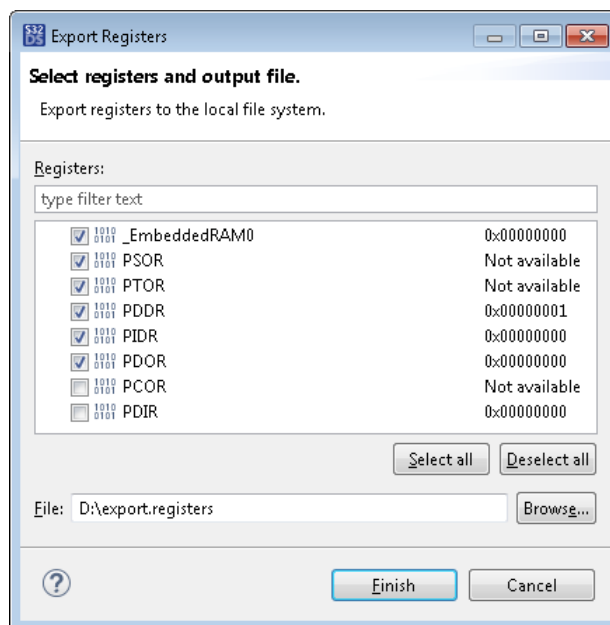
- **Import** a saved set of registers with values. For detailed information on exporting and importing registers refer to [Exporting register values](#) and [Importing register values](#) topics.

Exporting register values

When in a debug session, you can export values from registers to an XML-formatted file.

To export register values:

1. In the **Watch registers** view, click the  (Export registers to file) toolbar button on top of the **Watch registers** view or select registers, right-click and click **Export** on the context menu.
2. In the **Export Registers** dialog box, specify the full path of the file (TXT, XML, or other) for data export. If the file does not exist, it will be created during export.



3. Under **Registers**, choose to export all registers or leave the selected registers.
4. Click **Finish**.

The data is exported to the specified file in the XML format:

```
<registers>
  <register path="CSE_PRAM/_EmbeddedRAM0" value="0x00000000"/>
  <register path="PTE/PDDR" value="0x00000001"/>
  <register path="PTE/PDOR" value="0x00000000"/>
  <register path="PTE/PIDR" value="0x00000000"/>
  <register path="PTE/PSOR"/>
  <register path="PTE/PTOR"/>
</registers>
```

The register value is not exported to the file if:


- register is write-only,
- register marked **Read on demand** and value is not read,
- user terminated reading before export and value is not read.

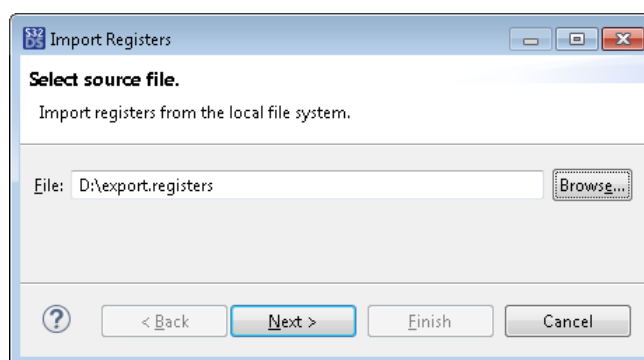
Importing register values

While in a debug session, you can import values to the **Watch registers** view from an XML-formatted file.

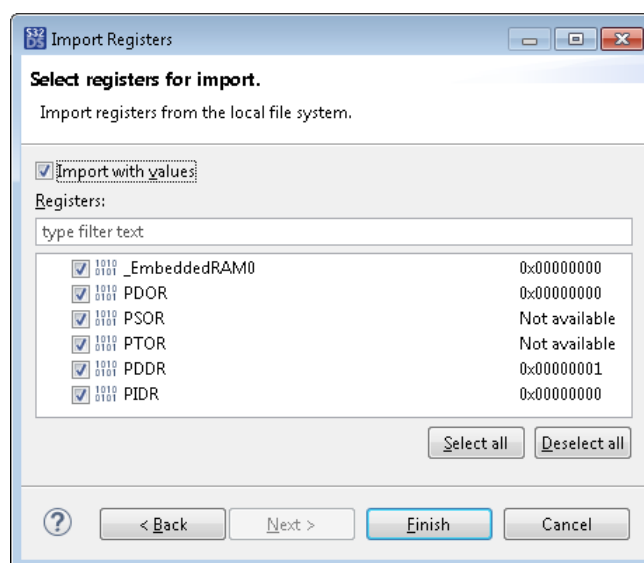
To learn how to get a file with register values for import, refer to topic [Exporting register values](#).

To import values to registers:

1. Open the [Watch registers view](#).
2. Click the  (Import registers from file) toolbar button on top of the **Watch registers** view.
3. In the **Import Registers** dialog box, specify the full path to the file containing the registers to be imported.



4. Click **Next**.
5. Choose to import only registers or registers with values.
6. Under **Registers**, choose to import all registers or leave the selected ones.



7. Click **Finish**.

If imported with values, the values are loaded to the registers. The **Watch registers** view displays the imported registers.

The value is not imported if:

- register is read-only,
- register is write-once or read-write-once and value is already set.

Such registers are skipped rather than updated.

For registers marked **Read on demand** import gives the following results:

- if the **Watch registers** view was clear before import, it displays the “?” as value for the registers,
- if that same registers were added to the **Watch registers** view earlier (before import), it will display:
 - the previously read value,
 - the “?” as value if the **Read** was not performed for that register.

Peripherals	Hex	Binary	Address	Description
CSE_PRAM/ EmbeddedRAM0	?	?	0x14000800	CSE PRAM 0 Register
BYTE_3 (bits 0-7)	?	?		Data byte 3 of Rx/Tx frame.
BYTE_2 (bits 8-15)	?	?		Data byte 2 of Rx/Tx frame.
BYTE_1 (bits 16-23)	?	?		Data byte 1 of Rx/Tx frame.
BYTE_0 (bits 24-31)	?	?		Data byte 0 of Rx/Tx frame.
PTE/PSOR	(write only)	(write only)	0x400FF104	Port Set Output Register
PTE/PTOR	(write only)	(write only)	0x400FF10C	Port Toggle Output Register
PTE/PDDR	0x00000000	00000000...	0x400FF114	Port Data Direction Register
PTE/PIDR	0x00000000	00000000...	0x400FF118	Port Input Disable Register
PTE/PDOR	0x00000000	00000000...	0x400FF100	Port Data Output Register

Viewing peripheral registers in EmbSys Registers

When in a debug session, use the [EmbSys Registers view](#) to access data in the peripheral registers of the connected target. Find the details in the following topics:

- [Reading values from peripheral registers](#)
- [Exporting peripheral register values](#)
- [Importing peripheral register values](#)

When out of a debug session, the **EmbSys Registers** view displays the information about the peripheral registers from the project settings. If the debug session was terminated, the **EmbSys Registers** view displays the last received register values.

The **EmbSys Registers** view can be shared by multiple projects. Each time you click a project in any view of S32 Design Studio for S32 Platform, the **EmbSys Registers** view switches to the context of the selected project and updates the information accordingly. To learn the details, refer to topic [Switching to a different context](#).

Reading values from peripheral registers

During debugging the **EmbSys Registers** view can read and display the actual values from the peripheral registers of the connected device. The data can be read from the registers of the RO (read-only) and WR (read-write) types. Only one register can be read at each debug action like a step, a resume, a stop, or a breakpoint.

To start reading a register, double-click the register name in the **EmbSys Registers** view. The registers selected for reading are marked with the “blue arrow” sign and with the register names displayed in green font.

Register	Hex	Bin	Reset	Access	Address	Description
DMA						Enhanced direct memory access co
DMA						Enhanced direct memory access co
▶ CR	- not read -		0x00000400	RW	0x40002000	Control Register
▶ ES	- not read -		0x00000000	RO	0x40002004	Error Status Register
▶ ERQ	- not read -		0x00000000	RW	0x4000200C	Enable Request Register
▶ EEI	- not read -		0x00000000	RW	0x40002014	Enable Error Interrupt Register
▶ CEEI	- write onl...		0x00	WO	0x40002018	Clear Enable Error Interrupt Register
▶ SEEI	- write onl...		0x00	WO	0x40002019	Set Enable Error Interrupt Register

To read the entire group of registers, double-click the group name. Similarly, double-click a register or a group of registers to stop reading their values.

Note: Be careful when selecting a register group with a large number of registers for reading as it might slow down the debugging. Also, you may encounter a problem with an abrupt disconnect occurring at the attempt of the client to access the selected group of registers. If this be the case, try to close the **EmbSys Registers** view and start a new debug session without this view.

When the **Hex** and **Bin** fields are updated with the actual values of the registers, these values are displayed in red font.

The “not read” message displayed in the **Hex** field means that the register value cannot be read. Make sure that the **EmbSys Registers** view uses the context of the project you are debugging. Learn more in topic [Switching to a different context](#).

If the “not read” issue occurs in the right context, it can indicate a register not initialized properly. To learn more, refer to the respective hardware manual. The manuals are available in folder <S32 Design Studio for S32 Platform installation path>/S32DS/help/resources/hardware.


Exporting peripheral register values

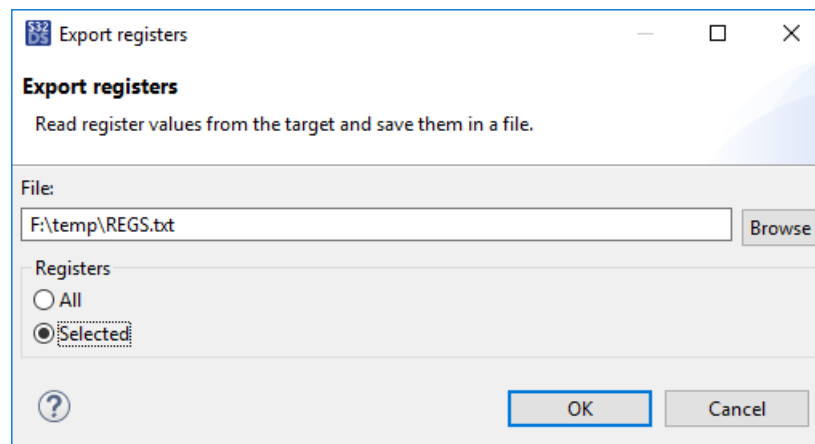
When in a debug session, you can export values from peripheral registers to an XML-formatted file.

To export peripheral register values:

1. In the **EmbSys Register** view, expand the required group of registers and double-click registers to start reading their values.

The registers selected for reading are marked with the “blue arrow” sign.

2. Click the  (Export selection to file) toolbar button on top of the **EmbSys Registers** view.
3. In the **Export registers** dialog box, specify the full path of the file (TXT, XML, or other) for data export. If the file does not exist, it will be created during export.



- Under **Registers**, choose to export all registers or only selected registers. A register or a group of registers is selected when clicked and highlighted in the **EmbSys Registers** view.
- Click **OK**.

The data is exported to the specified file in the XML format:


```
<?xml version="1.0" encoding="UTF-8"?>
<register-groups>
  <register name="MDMAPSTTS">
    <value>0xDFDFDFDF</value>
    <address>3</address>
    <description>MDM AP status register</description>
  </register>
  <register name="MDMAPCTL">
    <value>0xDFDFCFDF</value>
    <address>7</address>
    <description>MDM AP control register</description>
  </register>
  <register name="MDMAPWIREN">
    <value>0xDFDFDFDF</value>
    <address>51</address>
    <description>MDM AP WIR enable register</description>
  </register>
  <register name="MDMAPWIRREL">
    <value>0xDFDFDFDF</value>
    <address>59</address>
    <description>MDM AP WIR release register</description>
  </register>
  <register name="MDMDAPENCTRL">
    <value>0xDFDFDFDF</value>
    <address>83</address>
    <description>MDM AP DAP enable control register</description>
  </register>
  <register name="MDMDAPENST">
    <value>0xDFDFCFDF</value>
    <address>87</address>
    <description>MDM AP DAP enable status register</description>
  </register>
</register-groups>
```

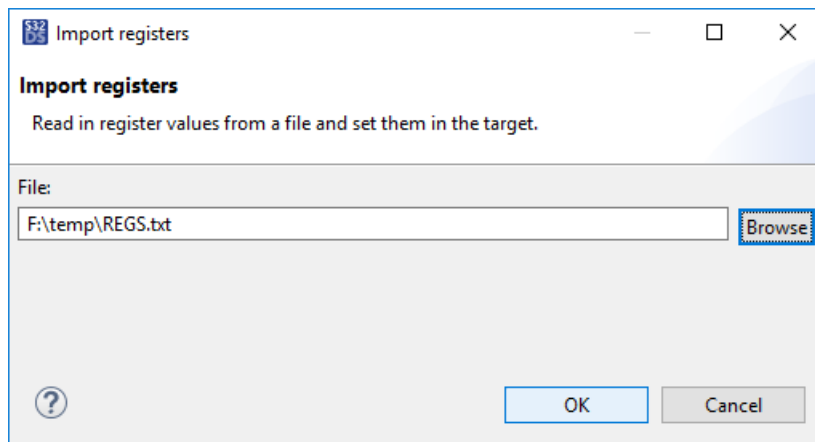
If a register is read-only or not marked for reading, the register value is not exported to the file. Instead, the “value” XML element shows the “?” character for each byte.

Importing peripheral register values

While in a debug session, you can import values to peripheral registers from an XML-formatted file. To learn how to get a file with peripheral register values for import, refer to topic [Exporting peripheral register values](#).

To import values to peripheral registers:

- Open the **EmbSys Registers** view.
- Click the  (Import from file) toolbar button on top of the **EmbSys Registers** view.
- In the **Import registers** dialog box, specify the full path of the file containing the register values to be imported.

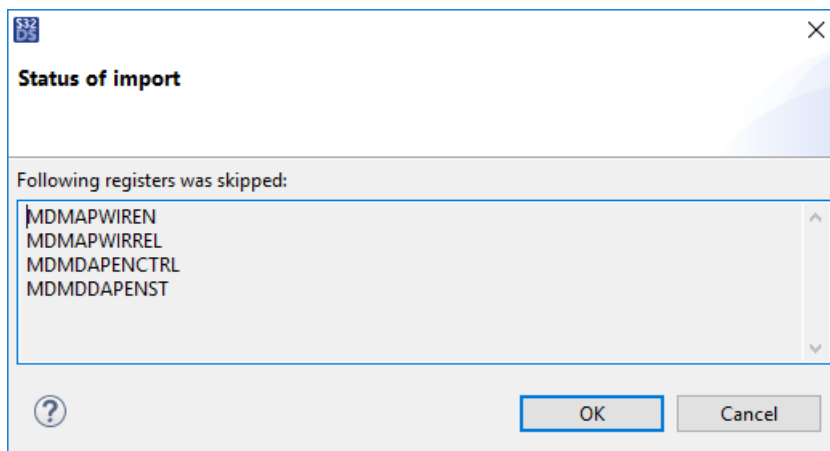


4. Click **OK**.

The imported values are loaded to the peripheral registers. The **EmbSys Registers** view displays the updated register values in red font.

Register	Hex	Bin	Reset	Access	Address	Description
> REG_PROT						REG_PROT
▼ DEBUG_CC						Subsystem
▼ DEBUG_CC						Subsystem
> MDMAPSTTS	0xDFDFDFDF	110111111101...	0x00000000	RO	0x00000003	MDM AP status register
> MDMAPCTL	0x00000000	000000000000...	0x00400000	RW	0x00000007	MDM AP control register
> MDMAPWIREN	0xDFDFDFDF	110111111101...	0x00000000	RW	0x00000033	MDM AP WIR enable register
> MDMAPWIRREL	0xDFDFDFDF	110111111101...	0x00000000	RW	0x0000003B	MDM AP WIR release register
> MDMDAPENCTRL	0xDFDFDFDF	110111111101...	0x00000000	RW	0x00000053	MDM AP DAP enable control register
> MDMDDAPENST	0xDFDFCFDF	110111111101...	0x00000000	RW	0x00000057	MDM AP DAP enable status register
> MUB						MUB

The values are imported to read-write registers only. Other registers are skipped rather than updated. You get a notification about all skipped registers:



Switching to a different context

The **EmbSys Registers** view displays the data from the context of a project that you are using at the moment. The name of this project and the related hardware information are displayed in the line above the data grid.

Note: If the line on top of the **EmbSys Registers** view shows the message “EmbSys Properties not set”, specify the required information in the project properties as described in topic [Setting up EmbSys properties for a project](#).

When a debug session is on, the **EmbSys Registers** view displays the registers of the project that you are debugging in the **Debug** view. If you switch to a different project, for instance, by clicking it in the **Project Explorer** view, then the **EmbSys Registers** view automatically loads the data from this last selected project. To get back to the context of the project you are debugging, click the **main()** node of the project in the **Debug** view.

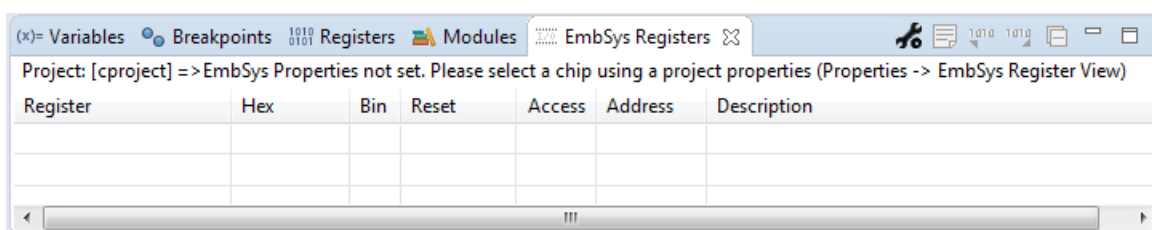
Note: The registers may not be read after a single click at **main()** if you get back to the **Debug** view from a different view. Click the **main()** node once again to start reading the registers.

Switching between the nodes in the **Debug** view also results in the loss of context in the **EmbSys Registers** view. If you switch from the thread of a running program to a different node such as the GDB client or other, the **EmbSys Registers** view stops reading the registers from the connected device. The **Hex** field starts displaying the “not read” message rather than the register values.

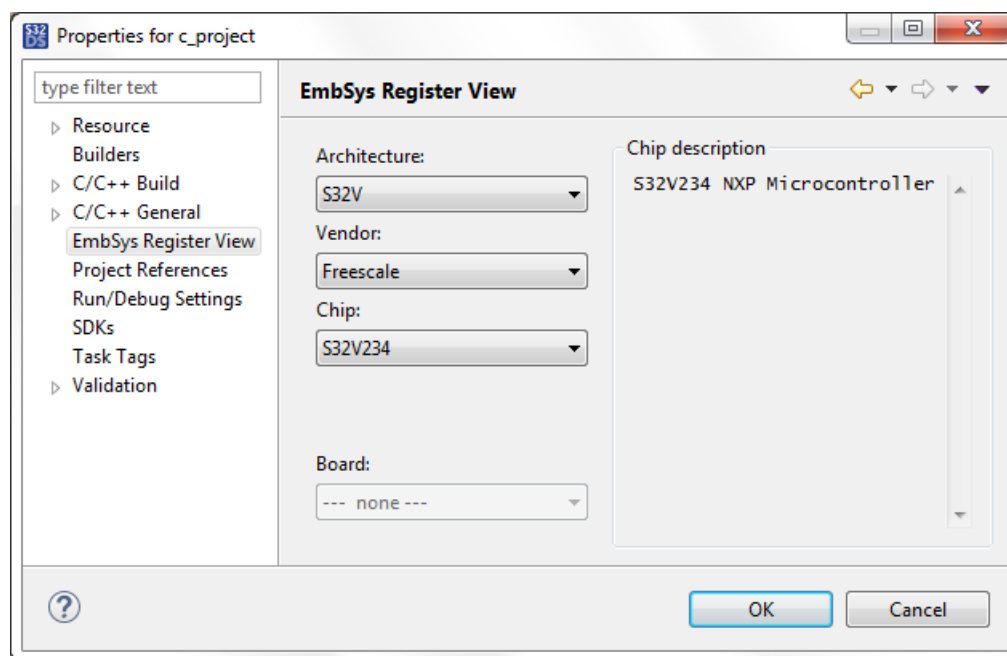
To restore the debug context, click the **main()** node of the project.

Setting up EmbSys properties for a project

If the **EmbSys Registers** view cannot receive information about the chip from the selected project, the line above the data grid displays the respective message:



The required settings can be specified in the **EmbSys Register view** section of the project properties.



The view responds to the changes in the project properties, including when the debug session is on.

Viewing memory

When in a debug session, you can monitor and modify your process memory in the **Memory view**. For this purpose, you can create memory monitors to view particular memory sections. A monitor is defined by a specific address (base address) or by an expression. By default, a monitor includes one memory rendering pane where you can examine the

contents of the memory section that starts from the base address. The rendered memory is displayed in one of the predefined formats: Floating Point, Traditional, Hex, ASCII, Signed Integer, Unsigned Integer, or Hex Integer.

If required, you can add more memory renderings to a memory monitor. Rendering can start with any valid address. Memory renderings are read-only.

Table 17: Actions allowed in the Memory view

Action	Steps
Add a memory monitor	<ol style="list-style-type: none"> 1. Right-click the Monitors pane and select Add Memory Monitor from the context menu. 2. Enter an address or an expression in the decimal or hexadecimal format. You can use the drop-down menu to select the previously specified expression. 3. Click OK.
Add rendering	<ol style="list-style-type: none"> 1. In the Monitors pane, select the memory monitor to which you want to add rendering. 2. Go to the New Renderings tab and click New Renderings. 3. Select a rendering type from the Select rendering(s) to create menu and click Add Rendering(s).
Remove rendering	In the Renderings pane, right-click the rendering pane and click Remove Rendering on the context menu. Or, click in the Renderings pane and then click the cross-sign icon on the toolbar in the Renderings pane.
Go to a specific address in a rendered memory section	<ol style="list-style-type: none"> 1. In the Renderings pane, right-click the rendering pane and click Go to Address... on the context menu. A group of controls appears in the Renderings pane. 2. Enter the required address in the blank edit box. 3. Click OK. The tab scrolls to the specified address.
Reset rendering to the base address	In the Renderings pane, right-click the rendering pane and click Reset to Base Address on the context menu. The tab scrolls to the base address.

Managing flash memory

If you need to take control of operations with flash memory during debugging, you can do it using the flash programmer tool delivered with S32 Design Studio for S32 Platform. This tool enables read-write access to QSPI flash memory of the connected target. All operations are performed from the command line.

To use the flash programmer:

1. Connect the target to your computer via USB or Ethernet.
2. Update the tool's configuration file as described in [Configuring a device connection](#).
3. Run the flash programmer from the console. Find the details in [Running the flash programmer](#).
4. Work with flash memory of the connected target as described in [Using commands](#).

Configuring a device connection

To configure a connection between the flash programmer and your flash device, open the `s32flash.py` script from the following location:

```
<S32DS 3.4 installation path>/S32DS/tools/S32Debugger/Debugger/scripts/gdb_extensions/flash/
```

Configure the following connection parameters, then save the file:

Table 18: Flash device connection parameters

Parameter	Description
<code>_FLASH_TYPE</code>	The type of flash memory.
<code>_PROBE_IP</code>	The connection with the debug probe. Options: <ul style="list-style-type: none"> “”: The probe is connected to the computer through USB. “<serial_number>”: Multiple probes are connected to the computer through USB, each identified by its serial number. “<IP> <host_name>”: The probe is connected to the computer using the Ethernet cable.
<code>_INIT_SCRIPT</code>	The script file used to initialize the target.
<code>_RESET_DELAY</code>	The delay at the beginning of the debug session.
<code>_FLASH_NAME</code>	The flash device name.

Running the flash programmer

Before you start, specify the python environment variable:

```
PYTHONPATH=<S32DS_install_dir>/S32DS/build_tools/msys32/mingw32/lib/python2.7
```

To run the flash programmer:

1. Navigate to `/S32DS/tools/S32Debugger/Debugger/Server/gta` and launch `gta.exe`.
2. Open the command prompt and run the file according to the device type:

Toolchain	Architecture	File	Location
GCC 6.3	32-bit Arm®	arm-none-eabi-gdb-py.exe	<code>/S32DS/build_tools/gcc_b1620/gcc-6.3-arm32-eabi/bin/</code>
GCC 9.2	32-bit Arm®	arm-none-eabi-gdb-py.exe	<code>/S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi/bin/</code>

3. In the GDB window, execute the following commands:

```
source <S32DS>/S32DS/tools/S32Debugger/Debugger/scripts/gdb_extensions/flash/s32flash.py
py flash()
```

Using commands

To manage flash memory, type the following commands in the command prompt. To get help on a certain command, type that command with the `-h` or `--help` parameter, for example, `fl_blankcheck -h`.

Table 19: Flash programmer - commands

Command	Description
<code>fl_blankcheck</code>	To check if flash memory is blank, use the following command: <pre>fl_blankcheck [-n {NUMBER all}] offset size</pre> where: <ul style="list-style-type: none"> • <code>offset</code>: Specifies the offset in the device's address range.

Command	Description
	<ul style="list-style-type: none"> • <code>size</code>: Specifies the size of flash memory (in bytes) to be checked. • <code>-n</code>: Specifies the number of mismatches to be shown in the console.
fl_dump	<p>Flash memory can be written to a binary file or output to the console. To dump a flash device, use the following command:</p> <pre>fl_dump offset size [-c {1, 2, 4, 8, 16} -f [FILE]]</pre> <p>where:</p> <ul style="list-style-type: none"> • <code>offset</code>: Specifies the offset in the device's address range. • <code>size</code>: Specifies the size of flash memory (in bytes) to be dumped. • <code>-f [FILE] (--file [FILE])</code>: Specifies the path of the binary file where the dump will be saved. The file path must not contain spaces. • <code>-c {1, 2, 4, 8, 16} (--cell {1, 2, 4, 8, 16})</code>: Specifies the number of bytes per cell. This option applies when the <code>-f</code> option is not used and the output is shown in the console. <p>For example, to dump flash memory to a binary file:</p> <pre>fl_dump 0x40000 0x20000 -f dump.bin</pre> <p>If the <code>-f</code> option is not used, the dump is displayed in the console. For example:</p> <pre>fl_dump 0x40000 0x20000</pre>
fl_erase	<p>To erase flash, use the following command:</p> <pre>fl_erase offset size</pre> <p>where:</p> <ul style="list-style-type: none"> • <code>offset</code>: Specifies the offset in the device's address range. • <code>size</code>: Specifies the size of flash memory (in bytes) to be erased. <p>For example:</p> <pre>fl_erase 0x40000 0x100</pre>
fl_erase_all	<p>To erase all flash memory, use the following command:</p> <pre>fl_erase_all</pre>
fl_protect	<p>To protect flash from erasing or reprogramming, use the following command:</p> <pre>fl_protect offset size</pre> <p>where:</p> <ul style="list-style-type: none"> • <code>offset</code>: Specifies the offset in the device's address range. • <code>size</code>: Specifies the size of flash memory (in bytes) to be protected. <p>For example:</p> <pre>fl_protect 0x100000 0x100</pre>

Command	Description
fl_current	<p>If multiple flash devices are connected to your computer, select a certain device using the following command:</p> <pre>fl_current dev</pre> <p>where dev specifies the name of the current device.</p>
fl_unprotect	<p>To make protected flash ready for erasing or reprogramming, remove protection using the following command:</p> <pre>fl_unprotect offset size</pre> <p>where:</p> <ul style="list-style-type: none"> • offset: Specifies the offset in the device's address range. • size: Specifies the size of flash memory (in bytes) from which protection will be removed. <p>For example:</p> <pre>fl_unprotect 0x100000 0x100</pre>
fl_info	<p>To view the details about the selected flash device, use the following command:</p> <pre>fl_info</pre>
fl_write	<p>To write a binary file or a hex value to flash memory, use the following command:</p> <pre>fl_write [-s [SIZE]] [--erase] offset data [--verify]</pre> <p>where:</p> <ul style="list-style-type: none"> • offset: Specifies the offset in the device's address range. • data: Specifies a hex value or a binary file that will be written to flash. The file path must not contain spaces. • -s [SIZE], --size [SIZE]: Specifies the size of flash memory (in bytes) to be written. • -e, --erase: Erases flash before writing new data. • -v, --verify: Verifies data written to flash. The data is compared with the file or value specified in the data parameter. <p>For example:</p> <pre>fl_write --erase 0x40000 u-boot.bin --verify</pre>
fl_write_elf	<p>To write an ELF file to flash memory, use the following command:</p> <pre>fl_write_elf [--erase][--verify][--base address] filename</pre> <p>where:</p> <ul style="list-style-type: none"> • -e, --erase: Erases flash before writing new data. • -v, --verify: Verifies data written to flash. The result is “OK” or “ERROR”. • -b, --base: Specifies the base address of the ELF file in flash memory. This option is required if the ELF file is built for the aliased region of flash memory.

Command	Description
	<ul style="list-style-type: none"> <code>filename</code>: Specifies the file name and the path at which the ELF file is located. Spaces are not allowed. <p>For example:</p> <pre>fl_write_elf --erase --verify --base 0x10000000 f:\test.elf</pre>

Debugging on multiple cores

This topic describes concurrent debugging of multiple applications running on different cores of the target and interacting with each other.

Designing embedded software for a multi-core target includes several steps as follows:


- **Step 1:** Create an application project for a multi-core target in S32 Design Studio. The project creation wizard generates several application projects, one per core.
- **Step 2:** Add custom code and compile each application project into an executable (ELF file). The executables are indexed for use on a particular core of the target. One of the executables (indexed “0”, or “0_0”, or “boot”) is intended for the boot core.
- **Step 3:** Debug the executables on the target. You can debug each executable as a standalone application, or you can load all executables to the intended cores and debug them in parallel.

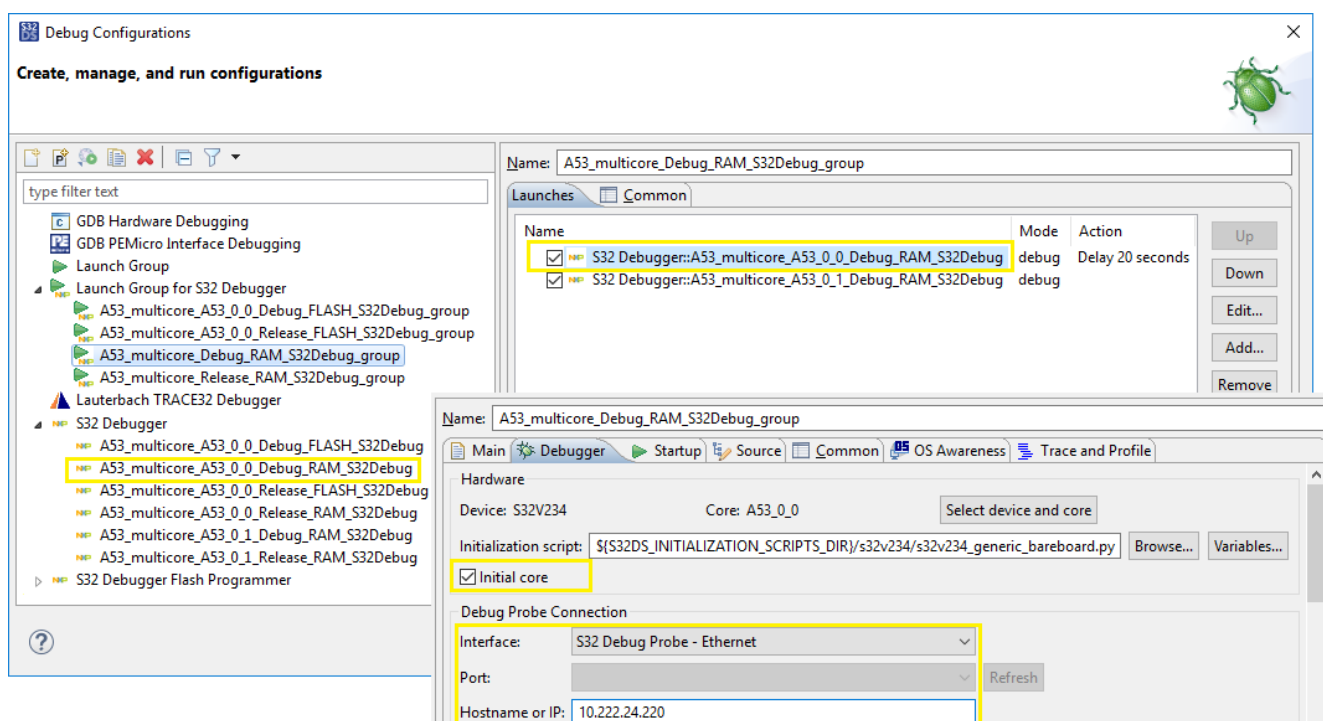
Note: Debugging on multiple cores in the simulation mode is not supported. Though you can run multiple debug sessions concurrently and switch between them, the simulator executes each application as a standalone process.

Loading executables for debugging on the target can be done manually or using a *launch group*. To load the code manually, run the first debug session for the boot core. If successful, add debug sessions for the remaining cores. Find the details in [Debugging on a bare-metal target](#).

When using a launch group, just run it. The settings inside the launch group specify the order of launching for the debug sessions and the time intervals between the launches. For details, refer to [Using launch groups](#).

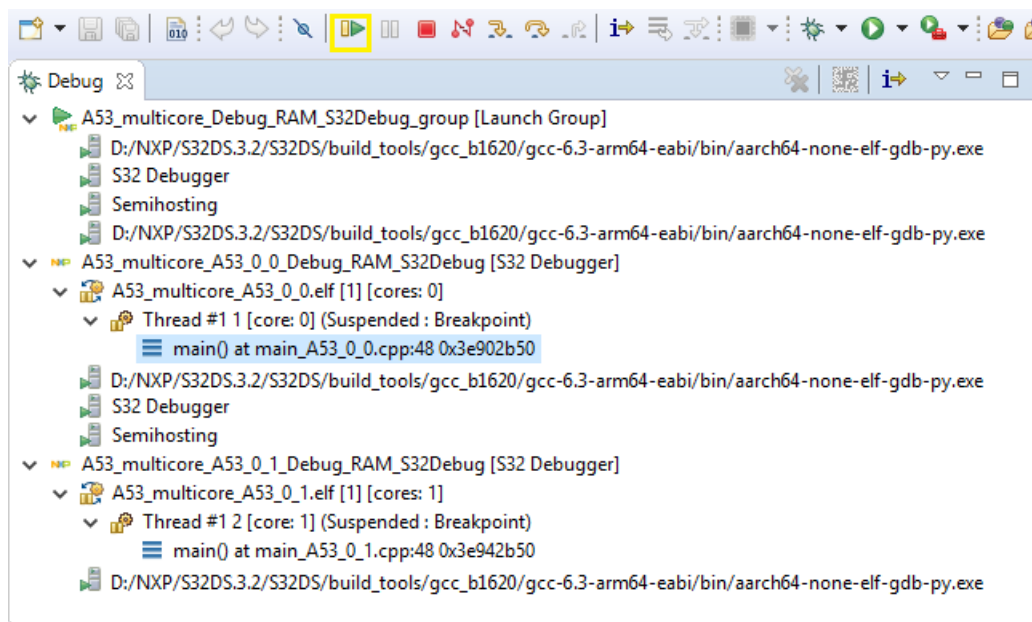
To start debugging on multiple cores using a launch group:

1. Click the  (*Debug As*) button on the toolbar, then click **Debug Configurations** from the drop-down menu.
2. In the **Debug Configurations** dialog box, expand **Launch Group** in the left pane and find the launch groups named as your project. Click the launch group created for debugging from FLASH or from RAM.
The right pane displays the included debug configurations in the order of launching. The top configuration will initialize the boot core and run the debug session for it.
3. In the left pane, expand the debugging interface (**S32 Debugger** or other) used by the board. Find inside the debug configuration intended for the boot core, and click it.




On the **Debugger** tab of the debug configuration, specify the board connection settings and make sure that the **Initial core** option is flagged. Click **Apply**.

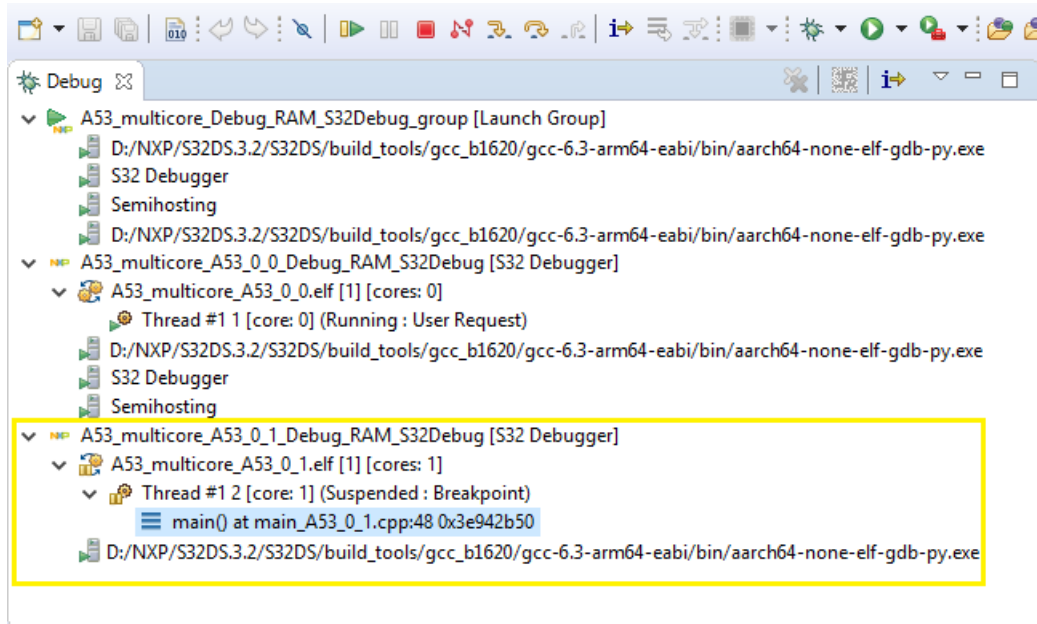
- Open the launch group again and click **Debug**. Wait for the debug sessions to be started. If prompted, confirm switching to the **Debug** perspective.
- When all debug sessions are started, they appear in the **Debug** view. If you did not modify the breakpoints in the debug configurations, all debug sessions are started and stopped at the first breakpoint (*main*).




To start debugging on the boot core, select the respective thread in the **Debug** view and click the **Resume** button on the toolbar, or press F8.


You can use debugging techniques such as stepping, breakpoints, stops, resumes, monitoring registers and memory, and other. To learn more, refer to [Using the debugger](#) and [Debugging on a bare-metal target](#).

- To continue debugging on a different core, switch to the respective thread in the **Debug** view. Click the  (*Resume*) button or press F8.

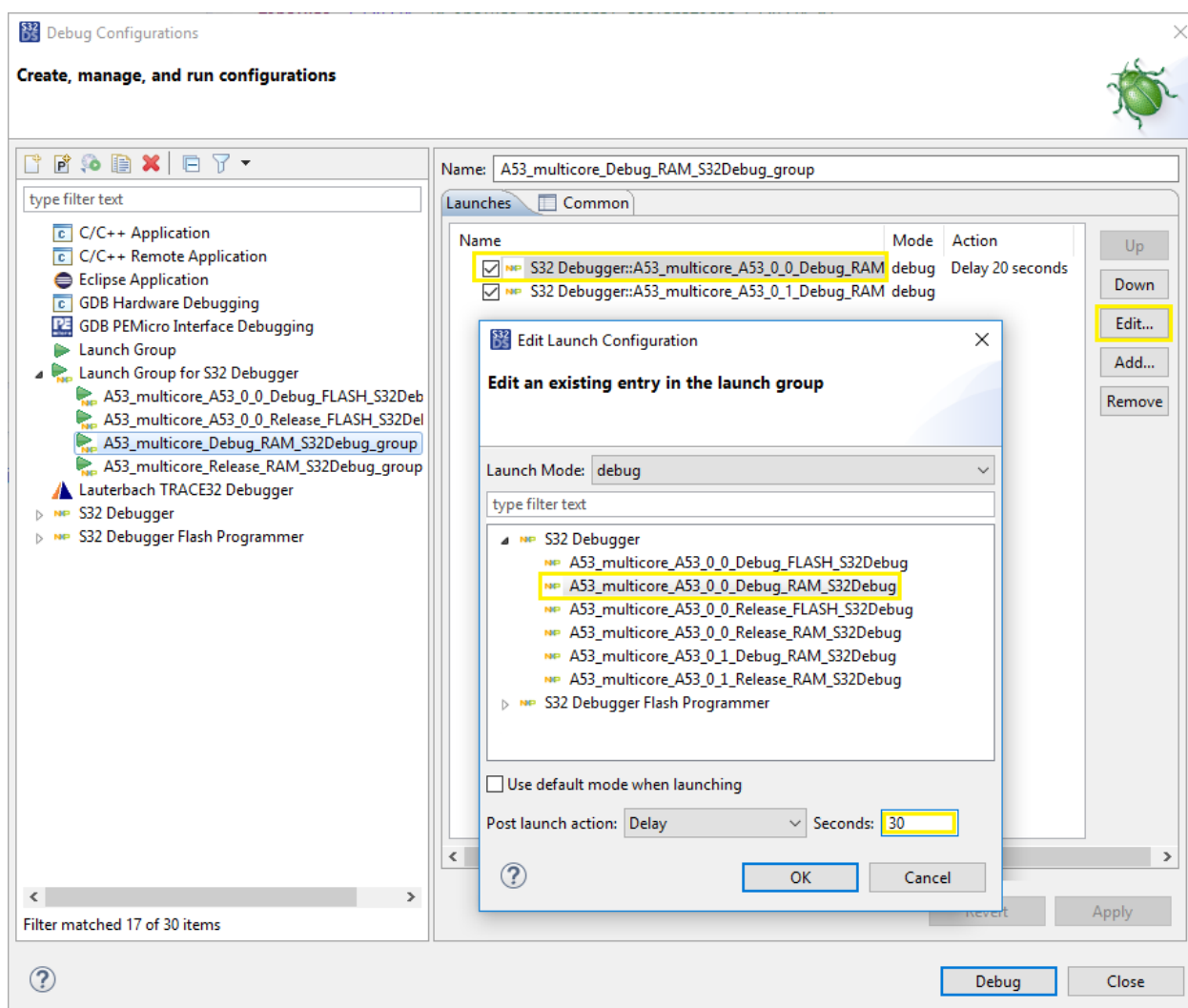


The other cores are up and running, executing their code and interacting with each other.

- When debugging is done, terminate the debug sessions. To terminate a debug session, select the respective thread in the **Debug** view, then click  (*Terminate*).

Termination of any debug session (primary or secondary) does not terminate the remaining debug sessions in a group. To terminate all debug sessions in a group at once, click the launch group, then click  (*Terminate*).

Running a launch group may sometimes end with unexpected termination of the secondary debug sessions. This may be caused by an attempt to launch the secondary debug sessions too early, when the boot core initialization is still on and the secondary cores are not up and ready yet. To solve this problem, open the launch group and click **Edit** for the first configuration.



In the **Edit Launch Configuration** dialog box and increase the delay. Alternatively, set the breakpoint right after the initialization section and select the **Wait for stop on breakpoint** post launch action. Click **OK** and **Apply**. Then try to run the launch group again.

Debugging on a Linux target

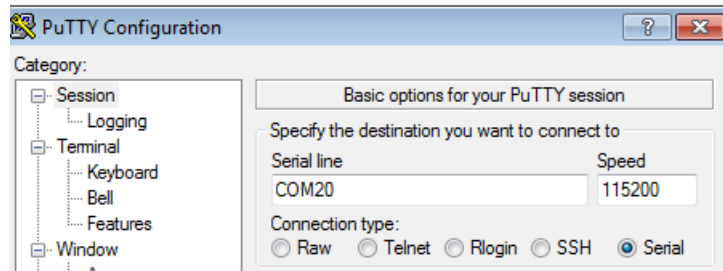
To debug an application on a target running Linux, you need an evaluation board with the target set up properly. Find the details in the documentation for the respective development package that you have installed on S32 Design Studio for S32 Platform in order to support the target.

Note: The user documentation is available in the help system of S32 Design Studio for S32 Platform. The PDF versions of the guides are located in folder `<S32DS_install_path>/S32DS/help/pdf`.

To debug an application on a target running Linux:

1. Connect the board to a USB port of your computer. Power up the board.
2. Build the application project.
3. Open the **Debug Configurations** dialog and go to the **C/C++ Remote Application** group of configurations. Click the launch configuration generated for your Linux application project.
4. On the **Main** tab, expand the **Connection** menu and select the connection to the Linux target. If you have not created this connection before, do it as follows:

- a. With the Linux running on the board, start a terminal program (for example, PuTTY) on your PC computer:



Set the connection type to “Serial”. Set the speed to “115200”, data bits to “8”, stop bits to “1”, and parity to “None”.

Then specify the destination you want the terminal to connect to, for instance, the USB port on your computer to which the board is connected.

- b. Start the terminal session and log into Linux (for example, use the “root” login name).
- c. To get the IP address of the Linux target, enter the following command:

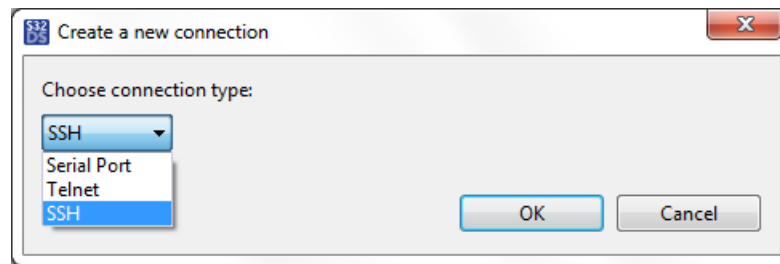
```
ifconfig
```

The output includes the section for Ethernet link. For example, this section can look as follows:

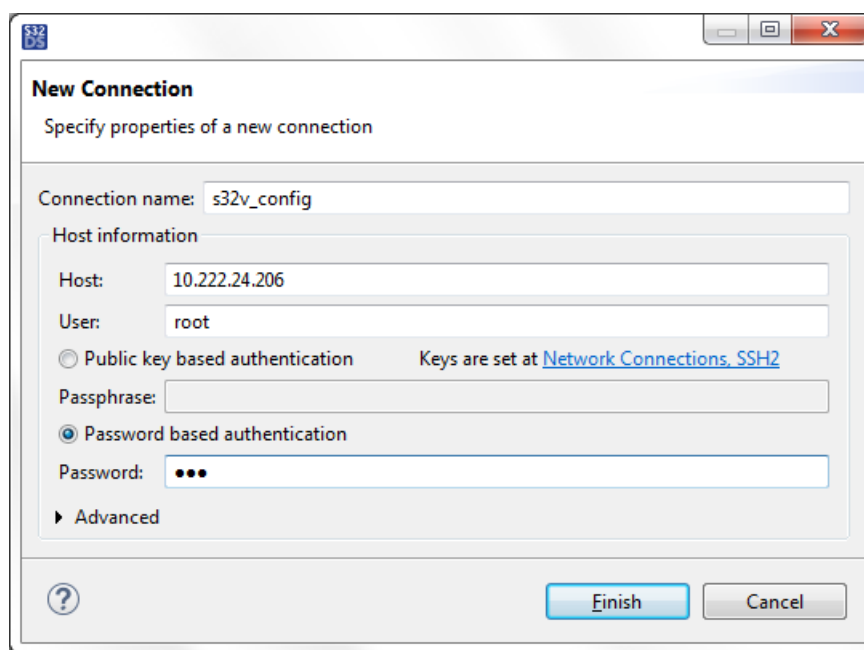
```
root@s32v234evb:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1b:c3:12:34:22
          inet addr:10.222.24.206  Bcast:10.222.24.255
          Mask:255.255.255.0
          inet6 addr: fe80::21b:c3ff:fe12:3422/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:709 errors:0 dropped:1 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:91479 (89.3 KiB)  TX bytes:1512 (1.4 KiB)
```

The *inet addr* parameter in this section is the IP address of the target.

- d. On the **Main** tab of the **Debug Configurations** dialog, click the **New** button next to the **Connection** field.
- e. In the **Create a new connection** dialog box, select **SSH** and click **OK**.



- f. In the **New connection** dialog box, specify the following settings:



- **Connection name:** Specify the preferred connection name.
 - **Host:** Enter the IP address of the target that you have obtained in the Linux terminal.
 - **User:** Enter the Linux user name (“root”).
 - **Password based authentication:** If required, enable authentication and enter the Linux user password.
- g. Click **Finish**.
5. On the **Debugger** tab, specify the following settings for GDB remote debugging:

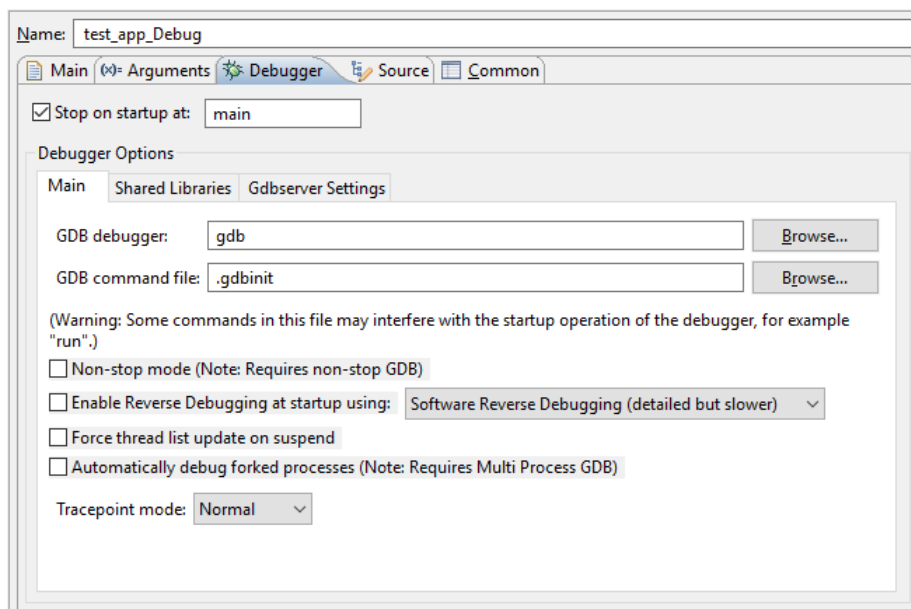


Table 20: Debugger tab: Settings for remote debugging

Setting	Description
Stop on startup at	Specify the location in code where the debugger will place the first breakpoint and stop. Default: main.

Setting	Description
Debugger Options: Main tab	<p>Specify the GDB debugger settings.</p> <ul style="list-style-type: none"> • GDB debugger: Specify the path of the GDB executable. • GDB command file: Specify the path of the GDBINIT file with the GDB commands to be executed at startup. • Non-stop mode: Select this option to enable non-stop debugging of multi-threaded programs. This mode enables the user to examine stopped program threads in the debugger while other threads continue to execute freely. • Enable Reverse Debugging at startup using: Select this option to start a debug session in the reverse debugging mode. Expand the menu and select hardware or software reverse debugging. • Force thread list update on suspend: Select this option for the thread list to be updated forcibly if the debug session is suspended. • Automatically debug forked processes: Select this option for the debugger to automatically debug child processes created with the fork function. • Tracepoint mode: Select the tracepoint mode. Options: Normal, Fast, Automatic.
Debugger Options: Shared Libraries tab	<p>Specify the paths on the target host where the GDB debugger will search for shared libraries with symbols. To adjust the priority of a search path, use the Up and Down buttons.</p> <p>Load shared library symbols automatically: Select this option to enable the GDB debugger to automatically find the local copy of the library and load its symbols unless the remote path of the respective library is specified in the list.</p>
Debugger Options: Gdbserver settings tab	<p>Specify the settings of the gdbserver program running on the target host.</p> <ul style="list-style-type: none"> • Gdbserver path: The gdbserver path on the target host. • Port number: The port for listening commands from the GDB host. • Gdbserver options: The command line options with which gdbserver is started.

6. Click **Apply**, then click **Debug**. The debug session is started and stopped at the first breakpoint.
7. Click **Resume** and debug the application.

Debugging on a VDK

This topic describes how to start a debug session in the simulation environment.

Before you proceed, make sure that you have installed the simulation software and performed the required configuration settings. For details, refer to **S32DS Installation Guide > Installing Synopsys tools**.

When using a certain VDK for the first time, add the VDK configuration files to the Synopsys workspace:

- In the S32 Design Studio installation directory, go to `/S32DS/config/vpconfigs/`. Copy the required S32x folder to the clipboard.
- In your Synopsys workspace directory, browse to `/NXP_S32xxxx_ECU/vpconfigs/` where the VDK configurations for the required target processor are located. Drop the copied S32x folder inside.

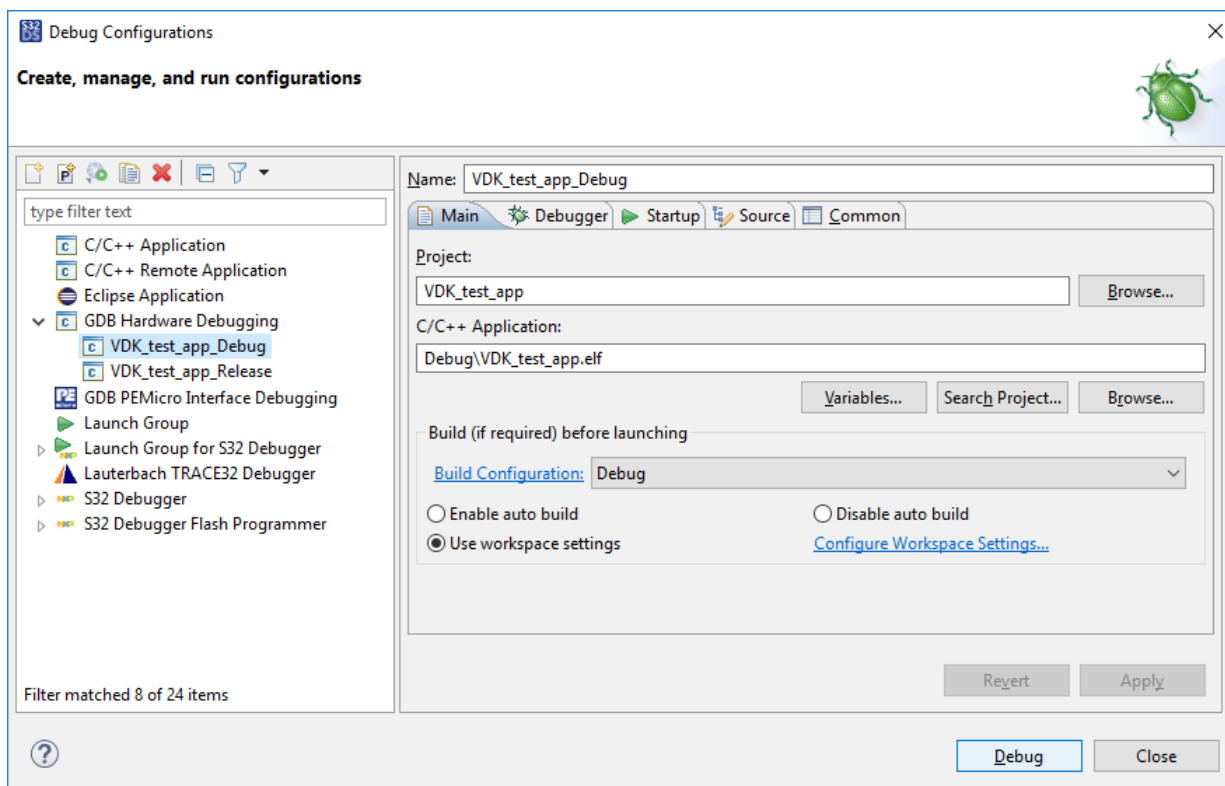
To run a debug session:

1. In S32 Design Studio, click **VDK Debug > Launch Simulation using VP Config** on the menu.

- In the **Open VP Config** dialog box, specify the following settings:
 - VP Project:** Locate the `snps.vpproject` file from the `<Synopsys_workspace_path>/NXP_S32xxxxx_ECU` folder.
 - VP Config:** Select the VDK configuration that you have copied to the Synopsys workspace.

Make sure that option **Launch simulation after opening VP Config** is selected.

- Click **OK** to start simulation. This procedure may take a few minutes.
- When the simulator has started, build the project.
- To start a debug session, click **Debug As > Debug Configurations**. In the **Debug Configurations** dialog box, go to **GDB Hardware Debugging** section in the left pane. Click the launch configuration generated for your project and click **Debug**.



- By default, simulation is suspended. To resume it, press **F8** or click **VDK Debug > Resume Suspended Simulation** on the menu.

If you need to debug a multi-core program, run launch configurations for all cores before clicking **Resume Suspended Simulation**. Notice that you can suspend execution of only one thread at a time during debugging.

Debugging Linux project on a VDK

This topic describes how to start a debug session for a Linux project in the simulation environment. Before you proceed, make sure that you have downloaded VDK, installed the simulation software and set the required environment variables. For details, refer to **S32DS Installation Guide > Installing Synopsys tools**.

When using a certain VDK for the first time, add the VDK configuration files to the Synopsys workspace: launch Virtualizer Studio with the *Run as administrator* option and select your target from the Fixed VDKs list on the Welcome page.

- Before simulating debug session on a Linux target for the first time, setup your environment:

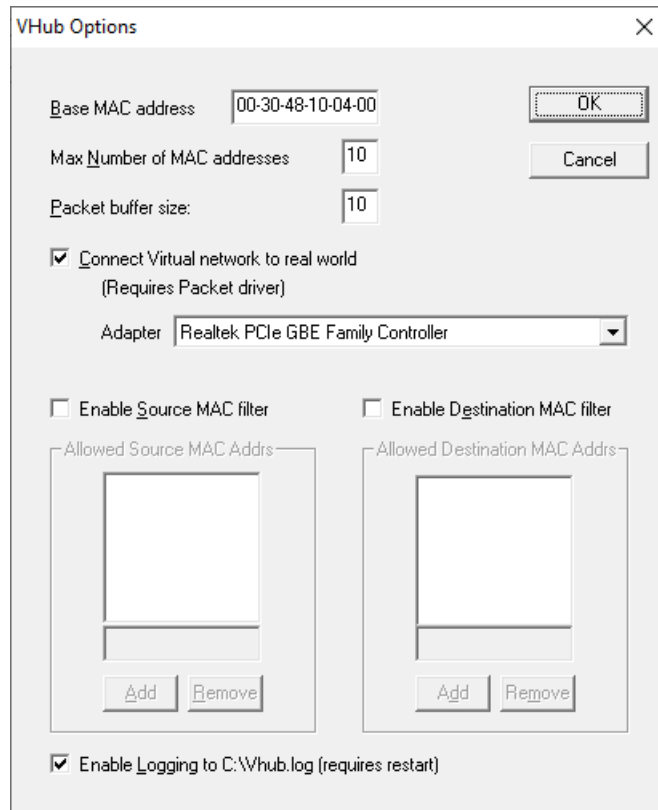
- a) Download Linux BSP from the Automotive SW - Linux product list on the nxp.com website.
- b) Copy the following files from the `s32<mcu>sim` folder to `vdk_workspace/NXP_S32<mcu>_ECU/software/s32xxxx_gen1/output`:
 - `fsl-image-auto-s32<mcu>sim.cpio.gz`
 - `Image`
 - `Image-s32<mcu>-simulator.dtb`
 - `u-boot-s32<mcu>sim.bin`
- c) Open the `vdk_workspace/NXP_S32<mcu>_ECU/vpconfigs/s32<mcu>_linux/s32<mcu>_linux.vpcfg` file and edit the last `paramOverrides` value to use `fsl-image-auto-s32<mcu>xsim.cpio.gz`:

```
value="{../../software/s32xxxx_gen1/output/fsl-image-auto-s32<mcu>xsim.cpio.gz,0x4000000,,image,} {../../software/s32xxxx_gen1/output/Image,0x80000,,image,} {../../software/s32xxxx_gen1/output/Image-s32<mcu>-simulator.dtb,0x2000000,,image,}"
```

- d) Install the VHub utility to support Real World I/O for Synopsys Ethernet models. The installer is located in the `vdk_workspace/NXP_S32<mcu>_ECU/bin/VirtualAndRealWorldIO/VHub` folder.
- e) Install the VHub Protocol driver. For details, refer to **VHub User Guide > Installing VHub on Windows** (`vdk_workspace/NXP_S32<mcu>_ECU/bin/Documentation/IPDocs/DESIGNWARE_ETHERNET/IP_VHubUserGuide.pdf`).

Now the environment is ready for debugging Linux project on a VDK.

2. Launch VHub with the *Run as administrator* option.

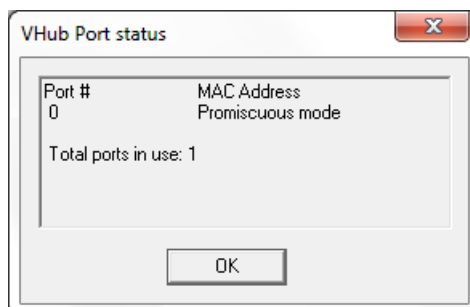



3. In Virtualizer Studio, click **VDK Debug > Launch Simulation using VP Config** on the menu bar.
4. In the **Open VP Config** dialog box, specify the following settings:

- **VP Project:** Locate the `snps.vpproject` file from the `vdk_workspace/NXP_S32<mcu>_ECU` folder.
- **VP Config:** Select the `s32<mcu>_linux` configuration.

Make sure that option **Launch simulation after opening VP Config** is selected.

5. Click **OK** to start simulation. This procedure may take a few minutes.
6. (Optional) When the simulator has stopped at the `initial_crunch` breakpoint, check the Vhub Port status:



7. By default, simulation is suspended. Click  (Resume suspended simulation) on the toolbar.
8. Go to the `LIN_MONITOR_0_B` terminal view and login as root. You can use the U-boot `run bootcmd` command.
9. Load the GMAC driver and obtain dynamic address:

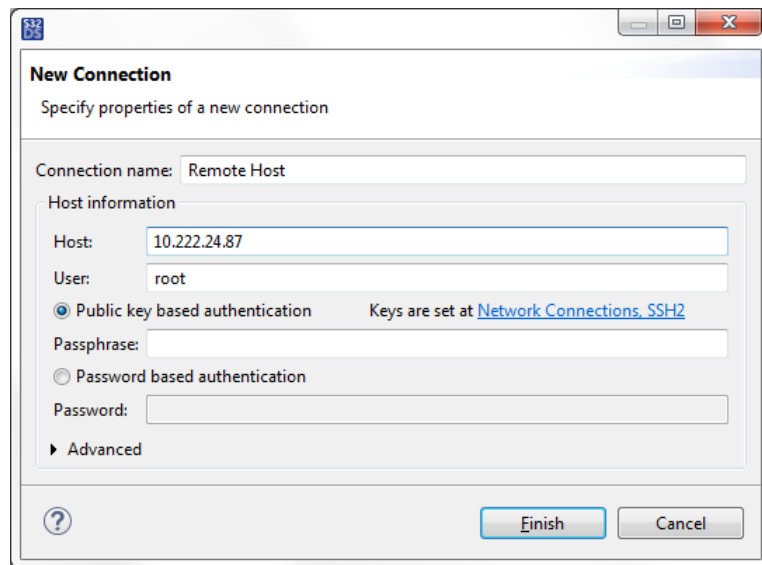
```
modprobe dwmac-s32cc
udhcpc -i eth0 -n -q
```

You can see the IP address in the Linux terminal and the VHub registration MAC addresses in Simulation Output.

10. Start the SSH server with the following commands:

```
/usr/bin/ssh-keygen -A
mkdir -p /var/run/sshd
/usr/sbin/sshd
```

11. In `Project Explorer`, use the project wizard to [create a new project](#) for a Linux target.
12. Build the project.
13. Open the Debug Configurations dialog and go to the C/C++ Remote Application group of configurations. Click the debug configuration for debugging remote Linux generated for your project.
14. On the Main tab, click the **New** button next to the Connection field to create a new remote connection. Then select the SSH connection type.
15. In the New connection dialog box, specify the following settings:



- Connection name: Specify the preferred connection name.
- Host: Enter the VDK IP address that you have obtained in the Linux terminal.
- User: Enter the Linux user name (root).
- Authentication: If required, enable authentication and enter the passphrase or password.

Then click **Finish**.

16. Click **Apply**, then click **Debug**.

Importing an executable

To debug a standalone executable file, you need to import it to a new or existing project in your workspace. This guide walks you through the process of importing an ELF file into a local project.

1. Launch the Import Executable wizard: go to **File > Import...** on the menu. In the **Import** dialog, click **S32 Design Studio for S32 Platform > C/C++ Executable as S32DS Project**, then click **Next**.
2. Choose the ELF file for import:
 - a) Click the **Browse** button to browse to the ELF file.
The **Select executable** field displays the full path of the ELF file. The gray field next to the **Browse** button displays the processor architecture recognized from the ELF file.
 - b) In the tree of processors, point the processor and the core that are being target in the selected ELF file.
The **Description** field displays the information about the selected processor and core.
 - c) Click **Next**.
3. Choose the project for import:
 - a) Click the **New Project Name** option to create a new project for import, specify the project name.
Alternatively, click the **Existing project** option to use an existing project for import, then click **Search...** and pick the project for import from the list.
 - b) Configure the creation of a launch configuration. If you do not need a new launch configuration for debugging, remove the **Create Launch Configuration** flag. Otherwise, select the debug configuration type from the list and specify the new configuration name.
 - c) Select from the list (if possible) the compiler to be used in the project.
The **Toolchain Name** field displays the selected compiler to be used in the project. Availability of toolchains depends on device and launch configuration selected.
 - d) (Optional) Click **Browse** and browse to a folder with the source code associated with the ELF file.
The folder will be included into the **Source Lookup Paths** list in the new launch configuration.

4. Click **Finish**.

SDK management

Overview

In addition to predefined (*contributed* or *external*) SDKs that are shipped with S32 Design Studio for S32 Platform, you can add custom SDKs and use them in projects. You can create an SDK from the existing C/C++ source files, import an SDK from an external storage, or load an SDK using its *descriptor*. Also, you have an option to export a custom SDK to an archive and make it available for import on other workstations.

Location and visibility

In S32 Design Studio for S32 Platform, you add a custom SDK to a particular workspace or to a certain project directly. When added to a workspace, the custom SDK becomes *global*, that is, available for use in all projects that use the compatible language, MCU, core, and toolchain. Global custom SDKs appear in the project creation wizard along with the predefined SDKs. The list of SDKs can be viewed in the user preferences and in the project settings of projects that are compatible with this SDK.

When added to a project rather than to a workspace, the SDK is *local*, that is, visible in this project only. SDKs added to a project are displayed in the project properties. If necessary, you can make a local SDK global at any moment.

When you add an SDK, it gets to the collection of SDKs available to a given scope of projects. An added SDK then needs to be *attached* to a project, after which the SDK files and resources become part of the project and can be included in the build.

SDK descriptor

S32 Design Studio for S32 Platform learns the details about an SDK from the SDK descriptor provided in the XML format. The SDK descriptor defines all information about the SDK such as its name and version, supported cores and toolchains, included source files and resources.

When you create an SDK, its descriptor is generated automatically and stored in the preferences (if the SDK is global) or in the project properties (if the SDK is local). When you export an SDK, the `sources.xml` file with the SDK descriptor is generated automatically and included in the archive. When you import or load an external SDK, this file is expected to be in the SDK root folder.

Using SDKs in a project

To use an SDK in a project, you need to attach it at the project creation or later. A project can use multiple SDKs, each attached to particular or all build configurations. The SDK configuration specifies which SDK files will be linked or copied to the project structure. Some SDKs are provided in the form of modules. Each module includes the SDK files specific for the particular project type or build configuration. So you do not need to attach the entire SDK and can select the required SDK modules only.

When you detach an SDK, copied files are not removed from the project's build configurations. You can detach one or multiple SDKs from a particular or all project build configurations.

The reason for detaching may be the necessity of editing the SDK structure, properties, or files. An SDK cannot be edited until detached from all projects.

Adding an SDK

This section describes the ways to add an SDK to S32 Design Studio for S32 Platform:

- If you have a C/C++ code, you can make it an SDK. For details, refer to [Creating an SDK](#).
- You can load an SDK using XML descriptor. For details, refer to [Loading an SDK](#).
- You can import an SDK from an external storage. For details, refer to [Importing an SDK](#).

- You can create a new SDK on the basis of the MCAL SDK. For details, refer to [Importing MCAL SDK](#).

Then, you can export your custom SDK to an archive to be reused in a different workspace. For details, refer to [Exporting an SDK](#).

Creating an SDK

In S32 Design Studio for S32 Platform, you can create an SDK from the C/C++ source files and resource files. The resulting SDK can be stored in the workspace and be available for use in many application projects. Or, the created SDK can be stored in a particular project and be available in the scope of this project only.

To create an SDK:

- Open the location where the SDK will be stored:
 - To add the SDK to the workspace, click **Window > Preferences** on the main menu. In the **Preferences** dialog box, go to **S32 Design Studio for S32 Platform > SDK Management**.
 - To add the SDK to a particular project, right-click the project in the **Project Explorer** and click **Properties** on the context menu. In the **Properties** dialog box, go to **SDKs**.

Note: If you choose to add your SDK to a project, the generated SDK descriptor includes the language, MCU, core, and toolchain properties that are specified in the project. When made global later, this SDK will be compatible with projects that have similar properties. When created global, an SDK does not specify the above properties and can be used in a project.

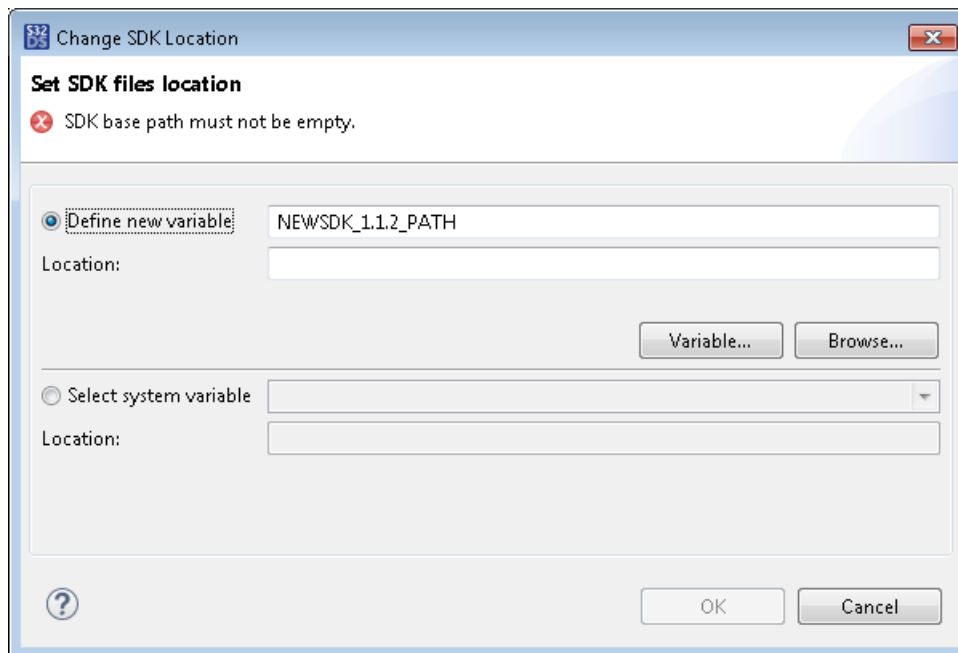
- Click the **Add...** button.
- In the **New SDK** dialog box, specify the SDK properties:

- Name:** Enter a valid name that starts with a letter. Use letters, digits, and underscores.
- Version:** Enter a string in the format “major.minor.micro.qualifier”. The “major” is mandatory, other parts can be skipped. The “qualifier” can include letters, digits and underscores, other parts can only use digits.
- Target folder name:** Enter a valid folder name that starts with a letter. After you attach the SDK to your project, the SDK files appear in the **Project Explorer** in the specified folder. Leave this field blank to use the SDK name for the project folder. This field is optional.

- **Description:** Enter a brief description of your SDK. This field is optional.

Note: The combination of the name and version must be unique in the workspace. This combination gives the name to the environment variable that is generated for the SDK automatically.

4. To set up the location of the SDK folder, click **Change...**
5. In the **Change SDK Location** dialog box, select the variable where the path of the SDK folder will be specified:



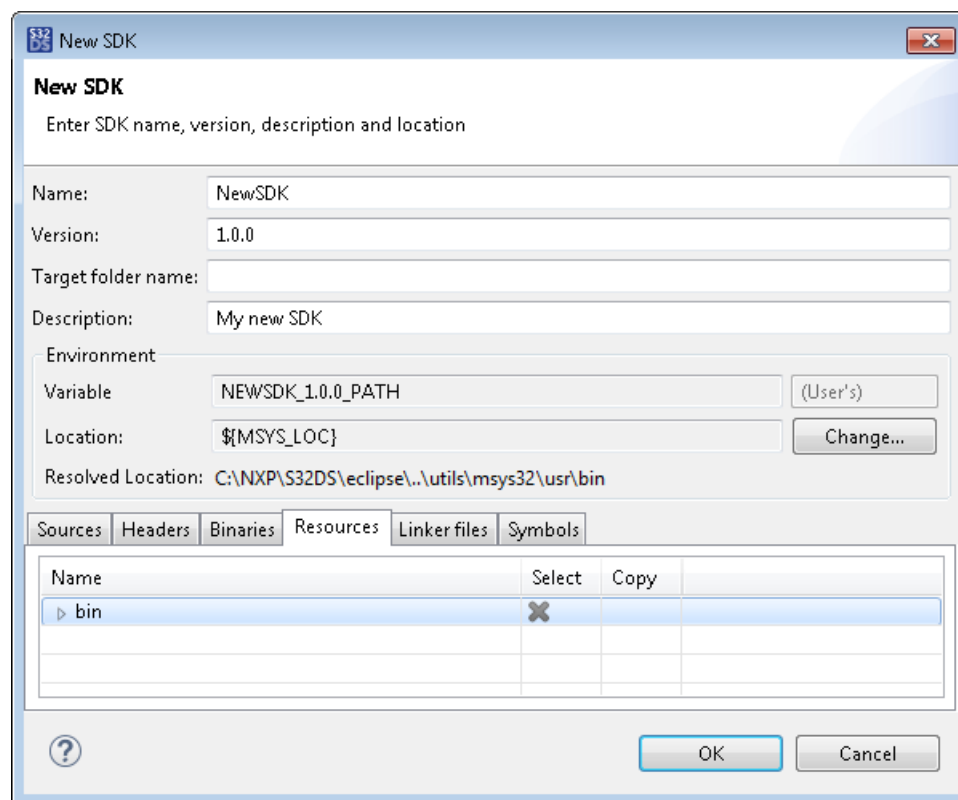
- **Define new variable:** Click to use a new variable for the SDK. To use the variable generated automatically, click **Browse** and browse to the SDK folder. Or, click **Variable** and select the environment variable that holds the path to the SDK folder.

Note: Use an environment variable to be able to share your SDK with other people or distribute it widely. If necessary, define a new environment variable and assign it with the SDK folder path in **Preferences > Run/Debug > String Substitution**.

- **Select system variable:** Click to use a system environment variable. Select the required variable in the drop-down list.

Once done, you can see the resolved location of the SDK folder.

6. Click **OK**. In the **New SDK** dialog box, select the files (source files, headers, binaries, resources, linker ID files) to be included in the SDK:



- In the **Select** column, mark the files with to be linked to a destination project.
- In the **Copy** column, mark the files with to be copied to a destination project.

By default, files are marked with (not selected).

7. Add compiler and preprocessor symbols on the [Symbols](#) tab.
8. Click **OK**.

Once done, the new SDK appears on the **SDK Management** page of the user preferences. If added to a project, the SDK also appears on the **SDKs** page of the project properties.

Loading an SDK

You can add an external SDK to your workspace using the SDK descriptor provided in the XML format. When added to a workspace, the loaded SDK becomes available for use in all projects with the compatible settings. The SDK source files are not copied to the product or workspace directory.

To load an SDK to your workspace:

1. Click **Window** > **Preferences** on the main menu. Then go to **S32 Design Studio for S32 Platform** > **SDK Management**.
2. Click the **Load...** button.
3. Browse to the folder where the SDK is located and select the XML descriptor.
4. Click **Open** to confirm the loading.

Once done, the loaded SDK appears on the SDK Management page.

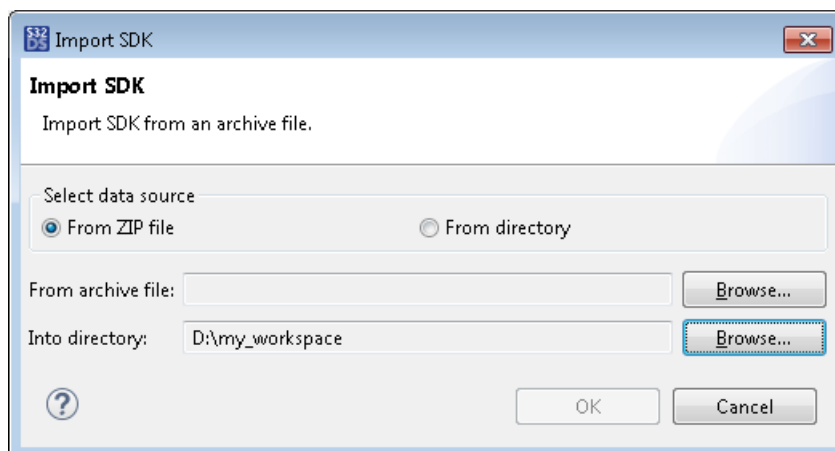
If there are some changes in original SDK after the loading, you can click the **Reload** button to be in sync with the latest updates. Alternatively, click **Remove** and use the **Load** button to add the updated version.

Importing an SDK

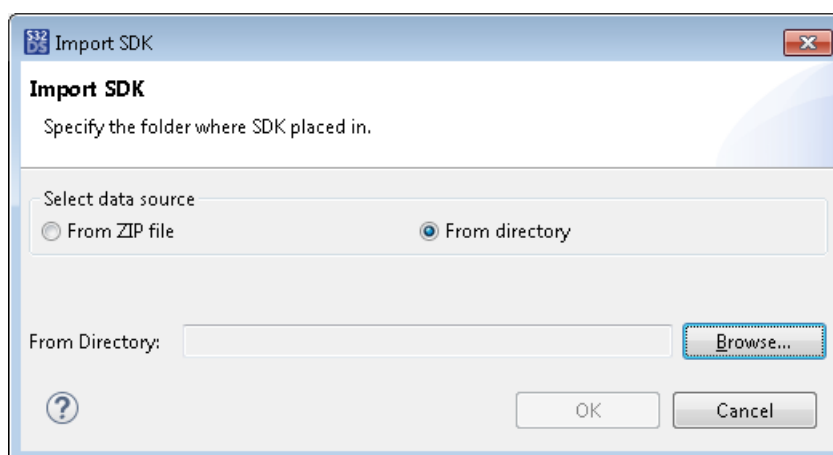
You can import an SDK to the preferred location (a workspace or a project) from an external storage such as a ZIP archive or a directory.

To import an SDK:

1. Open the location where to import the SDK:
 - To import the SDK to the workspace, click **Window > Preferences** on the main menu. In the **Preferences** dialog box, go to **S32 Design Studio for S32 Platform > SDK Management**.
 - To import the SDK to a particular project, right-click the project in the **Project Explorer** and click **Properties** on the context menu. In the **Properties** dialog box, go to **SDKs**.
2. Click the **Import** button.
3. In the **Import an SDK** wizard, click the source - a ZIP file or a directory.



4. Specify the import settings and confirm the operation:
 - To import from a ZIP file, browse to the archive file, then browse to the destination folder where the zipped files will be extracted. Click **OK**.
 - To import from a directory:
 - a. Browse to the folder where the SDK is located. Click **OK**.



- b. On the next wizard page, select the destination to which the SDK will be imported:
 - **Default (SDK folder):** The folder where the SDK is stored.
 - **SDK Base Path:** The standard base path to SDK files defined in the SDK descriptor.
- c. Click **OK**.

- (Optional) If an SDK with the same name and version already exists, specify different SDK properties in the **Change SDK name/version** dialog box and click **OK**.

Once done, the imported SDK appears on the **SDK Management** page of the user preferences. If imported to a project, the SDK also appears on the **SDKs** page of the project properties.

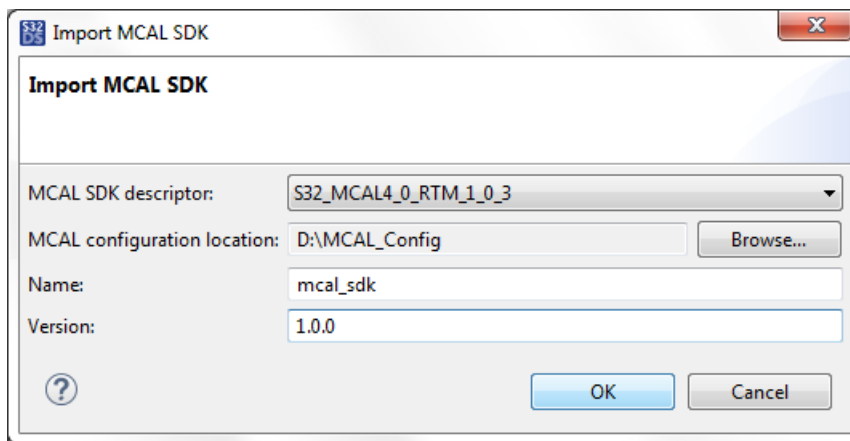
Note: If an SDK imported to a project does not appear on the **SDKs** page, this may be caused by incompatible properties or by unexpected or missing files in the SDK or project structure. Select the **Show All SDKs** option on the **SDKs** page and check if the imported SDK is marked accordingly. To learn more, refer to [SDKs](#).

Importing an MCAL SDK

Microcontroller Abstraction Layer (MCAL) is a software that provides direct access to the MCU modules and includes the microcontroller, memory, communication and I/O drivers. To use an MCAL SDK with device-specific tools in a project, import a respective MCAL configuration into a new SDK and attach it to the project.

To add an MCAL SDK:

- Click **Window > Preferences** on the main menu. In the **Preferences** dialog box, go to **S32 Design Studio for S32 Platform > SDK Management**.
- Click the **Import MCAL SDK** button.
- Select the descriptor from the **MCAL SDK descriptor** drop-down menu.
- Specify the location of the MCAL configuration files.
- Specify a unique SDK name that starts with a letter. Use letters, digits, and underscores.
- Specify the SDK version in the “major.minor.micro.qualifier” format. The “major” is mandatory, other parts can be skipped. The “qualifier” can include letters, digits and underscores, other parts can only use digits.



- Click **OK**.

The new SDK based on the selected MCAL configuration appears in the list of available SDKs. It is global and can be attached to any application project that uses the compatible language, MCU, core, and toolchain.

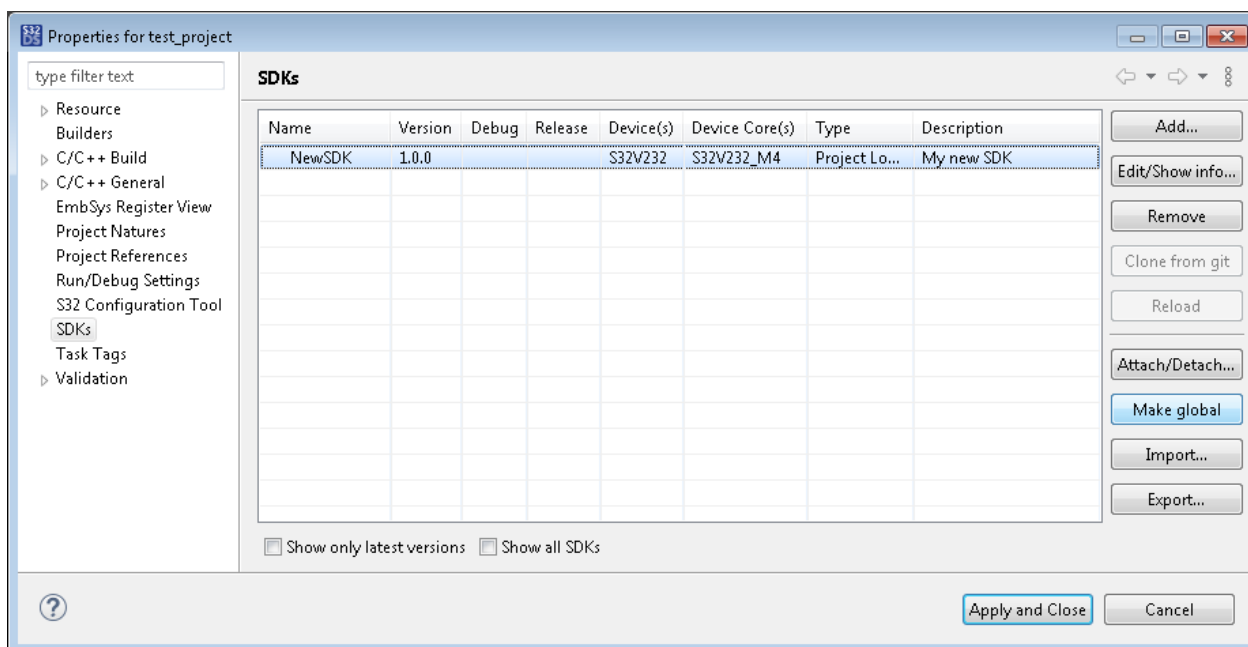
Making a local SDK global

An SDK added to a particular project can be used but locally. You can add a local SDK to the workspace and thus make it global.

When you add a local SDK to the workspace, the SDK becomes global but its descriptor still keeps the project's language, MCU, core, and toolchain. This global SDK can be used with projects that have similar properties.

To make a local SDK global:

1. Open the project properties by right-clicking the project in the **Project Explorer** and clicking **Properties** on the context menu.
2. In the **Properties** dialog box, go to **SDKs**.
3. Select the SDKs that you need to add to the workspace. Click **Make global**, then click **OK**.



The SDK becomes available on the **SDK Management** page of user preferences. Within the workspace, this SDK can be attached to any application project that uses the compatible language, MCU, core, and toolchain.

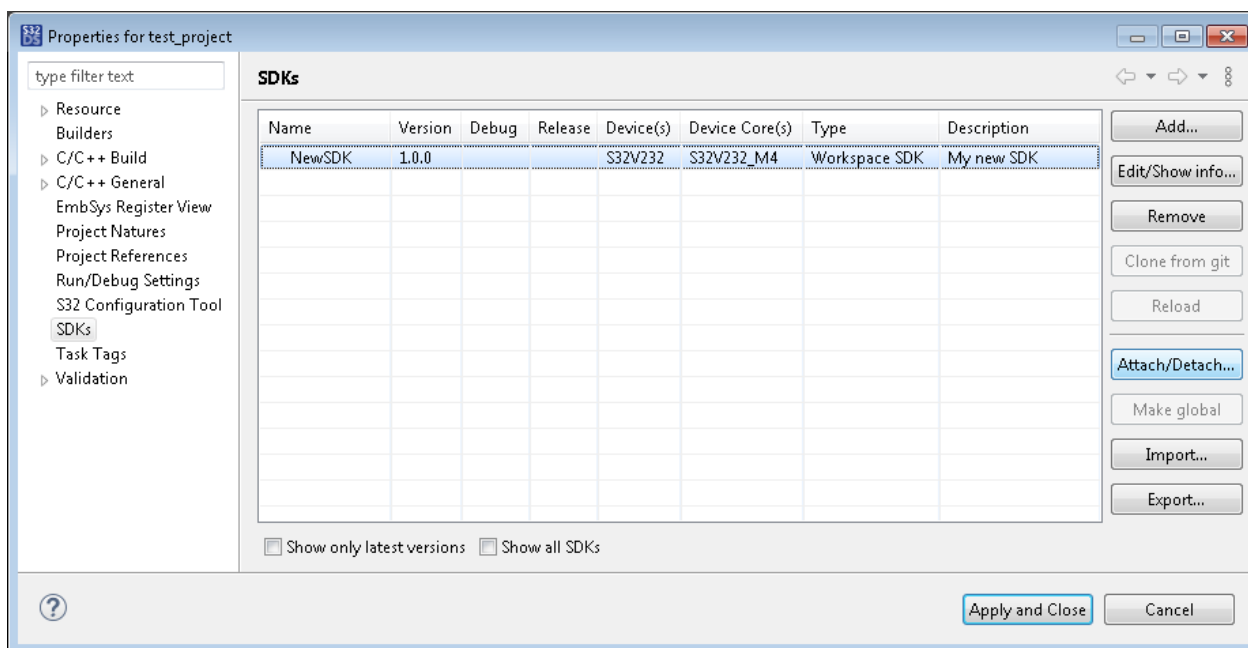
Using SDKs in projects

This section describes how to use an SDK in an application project. Find the details in the following topics:

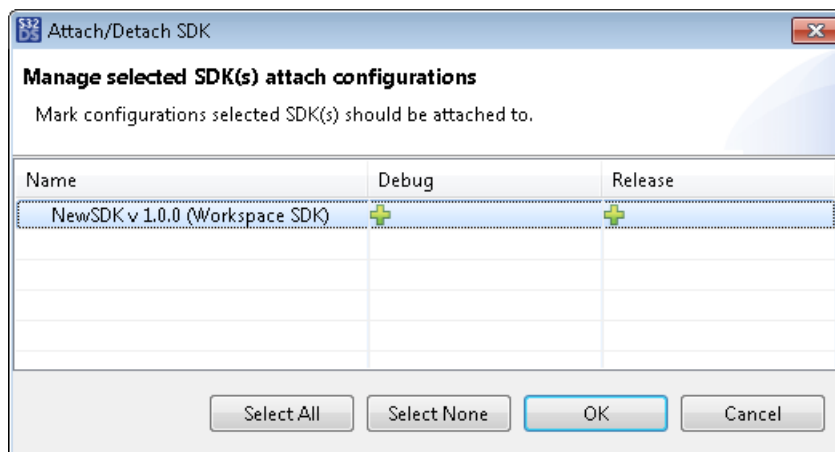
- [Attaching an SDK when creating a project](#)
- [Attaching an SDK to an existing project](#)
- [Upgrading an SDK version](#)
- [Detaching an SDK](#)

Attaching an SDK when creating a project

When creating a new project, you have an option to add an SDK on the second page of the wizard. Click the ellipsis button in the **SDKs** field. In the **Select SDK** dialog box, flag SDKs to be selected from the list and click **OK**.



- For each selected SDK, choose the build configurations to which it will be attached. Click **OK**.



- To complete the operation, click **OK** on the **SDKs** page of the project properties.

Upgrading SDK version

If the SDK you use in your project has a new version, you can migrate the project to use the latest available SDK.

To upgrade an attached SDK use the [Migrate wizard](#).

To learn the details, refer to topic [Migration guide](#).

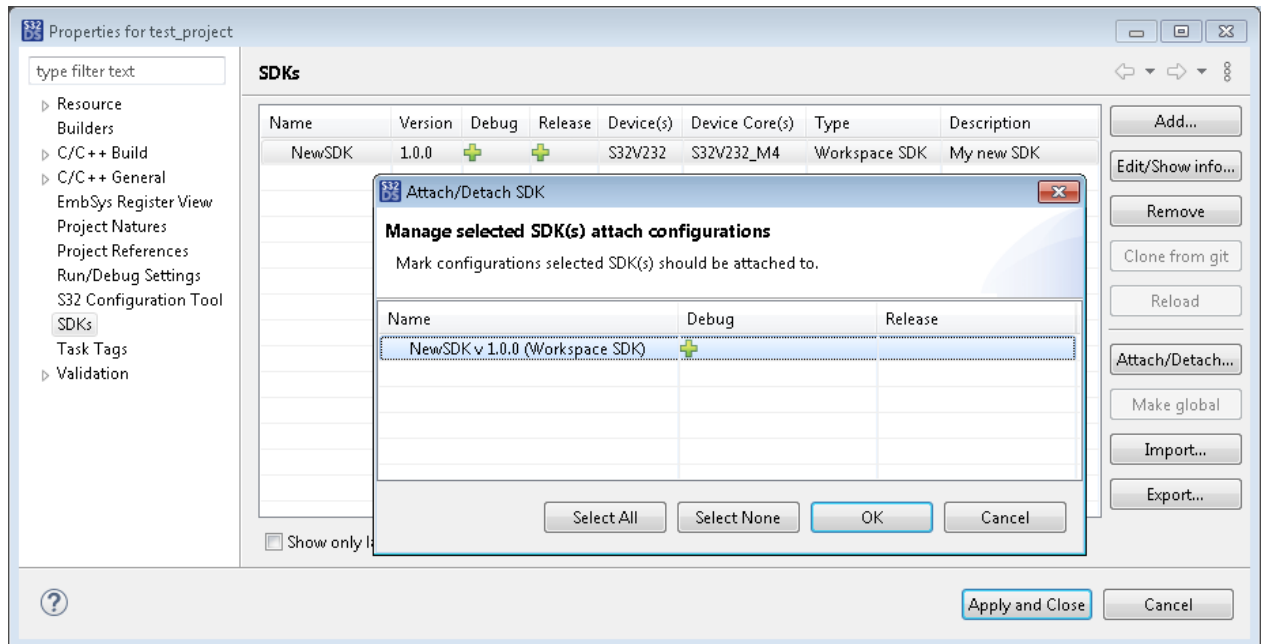
Detaching an SDK

An SDK can be detached from a particular project's configuration or from all project build configurations.

To detach an SDK:

- Right-click the project in the **Project Explorer** and click **Properties** on the context menu. In the **Properties** dialog box, go to the **SDKs** page.
- Select one or several SDKs to be detached and click the **Attach/Detach** button.

- Remove the **+** mark from the project build configurations from which the selected SDKs will be detached.



- Click **OK**. The detached build configurations are displayed without the **+** mark in the list.
- To complete the operation, click **OK** on the **SDKs** page of the project properties.

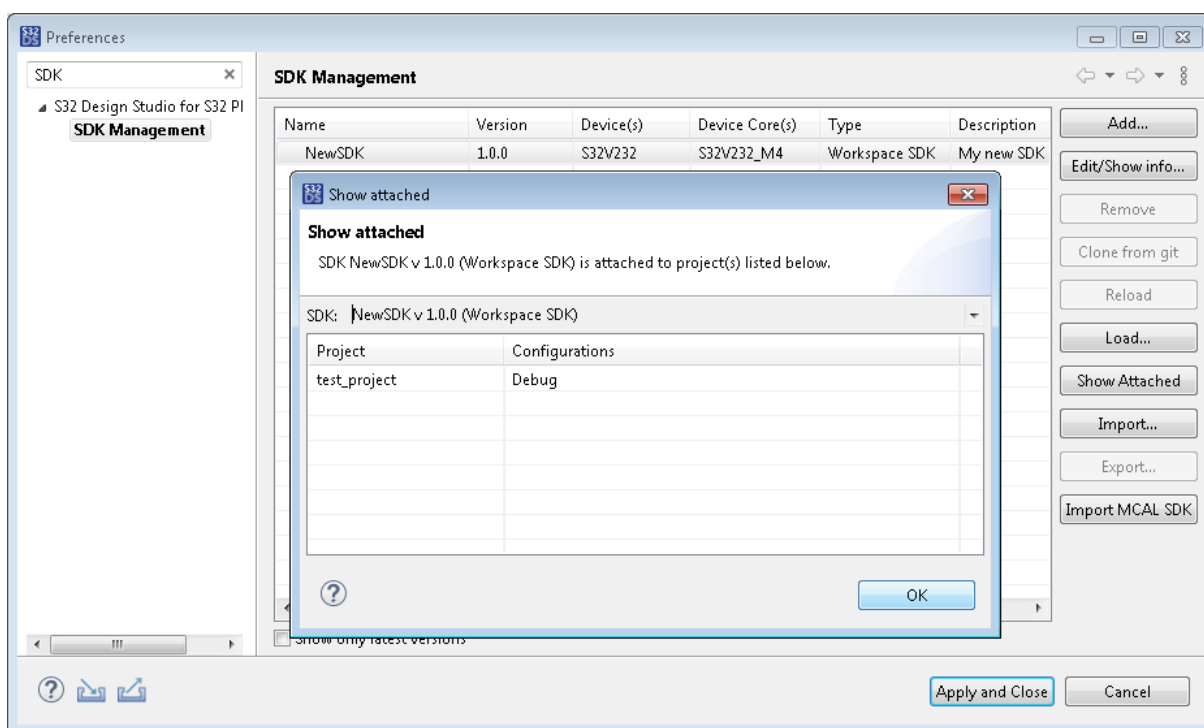
Editing an SDK

You can edit the properties and structure of a custom SDK that is not attached to any project.

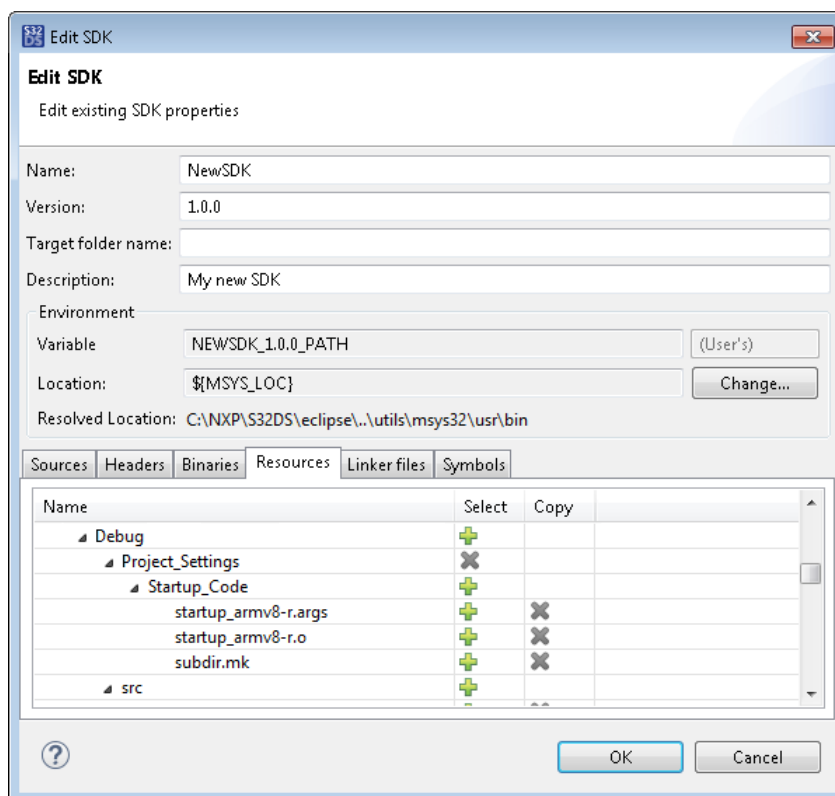
To edit an SDK:

- Go to the location where the SDK is available:
 - To open the list of SDKs in the workspace, click **Window > Preferences** on the main menu. In the **Preferences** dialog box, go to **S32 Design Studio for S32 Platform > SDK Management**.
 - To open the list of SDKs in the project, right-click the project in the **Project Explorer** and click **Properties** on the context menu. In the **Properties** dialog box, go to **SDKs**.
- Before editing an SDK, ensure that it is not attached to any project:
 - In the project properties, make sure that the **SDKs** page displays the local SDK without the **+** marks in all build configurations.
 - In the preferences, select the SDK on the **SDK Management** page and click **Show Attached**.

In the **Show attached** dialog box, select your SDK from the **SDK** drop-down menu and view all project build configurations to which the SDK is currently attached. Click **OK**.



3. If applies, detach the SDK from all project build configurations. For details, refer to topic [Detaching an SDK](#).
4. Get back to the location where the SDK is available. Click **Edit/Show info**.
5. In the **Edit SDK** dialog box, edit the SDK properties, linked and copied files, and defined symbols. For details, refer to topic [Creating an SDK](#).

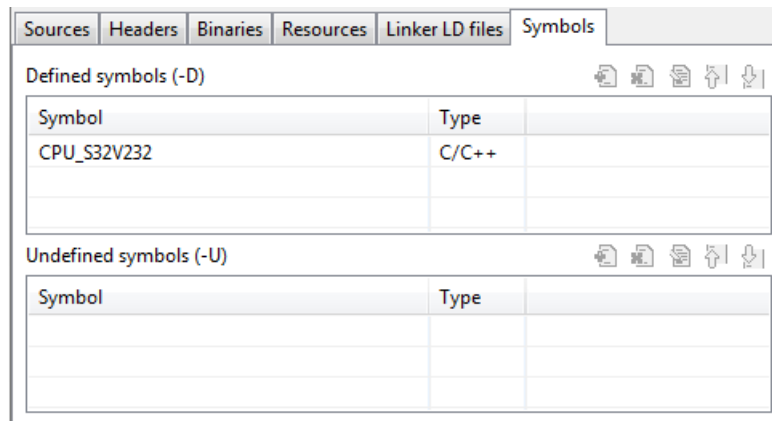


- Click **OK**.

Defining symbols

SDK provides ability to define symbols as macros with the `-D` option. These defined symbols will be added for the preprocessing tools after attaching SDK to the project.

- External (Contributed) SDK: the defined symbols list is read-only. Select SDK from the list on the **SDK Management** page, click **Edit/Show info...** and open the **Symbols** tab.
- Custom SDK: you can define symbols while creating a new SDK. If you need to specify the symbol type, edit the SDK descriptor. When you export an SDK, the `sources.xml` file with the SDK descriptor is generated automatically and included in the archive. Open this file, make changes and import the updated version.



There are four types of symbols:

- Common symbols. This type of symbols is declared for both the C/C++ Compiler and Assembler tools. Common symbols can be defined in two ways:
 - as comma-separated list in the `symbols` attribute of the `sdk` element

```
<sdk ... symbols="CPU_S32K144HFT0VLLT1, CPU_S32K144HFT0VLLT2" />
```

- as comma-separated list in the `value` attribute of the `commonSymbols` element

```
<sdk ...>
  <commonSymbols value="CPU_S32K144HFT0VLLT1,
    CPU_S32K144HFT0VLLT2" />
  ...
</sdk>
```

You can choose the most convenient way or use both, because the resulting list of the defined symbols will include all of them except duplicates.

- Symbols for the C Compiler tool. This type of symbols is declared only for the C Compiler tool as comma-separated list in the `value` attribute of the `cCompilerSymbols` element

```
<sdk ...>
  <cCompilerSymbols value="CPU_S32K144HFT0VLLT3, CPU_S32K144HFT0VLLT4" />
  >
  ...
</sdk>
```

- Symbols for the C++ Compiler tool. This type of symbols is declared only for the C++ Compiler tool as comma-separated list in the `value` attribute of the `cppCompilerSymbols` element

```
<sdk ...>
```

```
<cppCompilerSymbols value="CPU_S32K144HFT0VLLT5" />
...
</sdk>
```

- Symbols for the Assembler tool. This type of symbols is declared only for the Assembler tool as comma-separated list in the value attribute of the `assemblerSymbols` element

```
<sdk ...>
  <assemblerSymbols value="CPU_S32K144HFT0VLLT6, CPU_S32K144HFT0VLLT7" />
  >
  ...
</sdk>
```

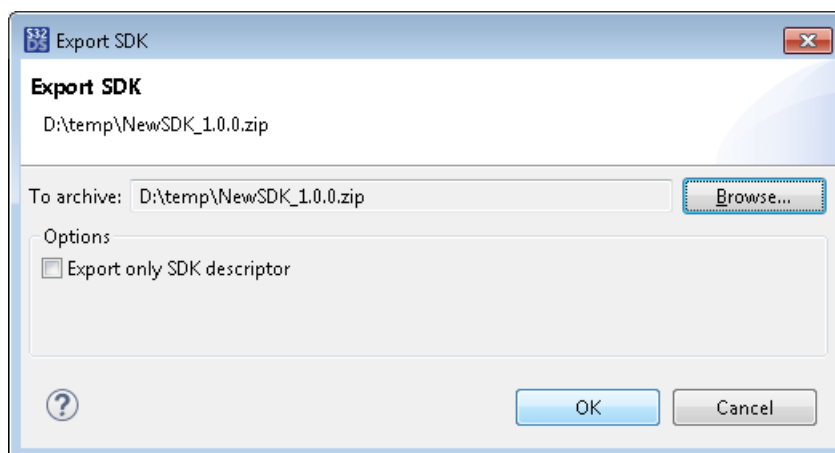
Exporting an SDK

You can export an SDK from the specified location (a workspace or a project) to a ZIP file. The resulting archive file will include all SDK files and the generated `sources.xml` file with the SDK descriptor. You have an option to export the `sources.xml` file alone, without the SDK files. For instance, you may find it useful if the SDK files are stored in a shared repository.

Note: You cannot export predefined SDKs and custom SDKs that were imported before.

To export an SDK:

1. Open the location where the SDK is available:
 - To export the SDK from the workspace, click **Window > Preferences** on the main menu. In the **Preferences** dialog box, go to **S32 Design Studio for S32 Platform > SDK Management**.
 - To export the SDK from a particular project, right-click the project in the **Project Explorer** and click **Properties** on the context menu. In the **Properties** dialog box, go to **SDKs**.
2. Click **Export**.
3. In the **Export SDK** dialog box, click **Browse** and browse to the folder where the ZIP file with the SDK name will be stored.



4. To export the `sources.xml` file only, select the **Export only SDK descriptor** option. The generated file will contain no references to the SDK files.
5. Click **OK**.

Removing an SDK

You can permanently delete one or multiple custom SDKs from the workspace. When you delete a custom SDK, it is automatically detached from all projects where it was used.

To delete a custom SDK:

1. Open the location where the SDK is available:
 - To open the list of SDKs in the workspace, click **Window > Preferences** on the main menu. In the **Preferences** dialog box, go to **S32 Design Studio for S32 Platform > SDK Management**.
 - To open the list of SDKs in the project, right-click the project in the **Project Explorer** and click **Properties** on the context menu. In the **Properties** dialog box, go to **SDKs**.
2. Select one or several SDKs and click **Remove**.
3. In the **Remove SDK confirmation** dialog box, click **OK** to confirm the removal.

Migration guide

You can use the [Migrate wizard](#) to convert SDK or GCC based toolchain for your project.

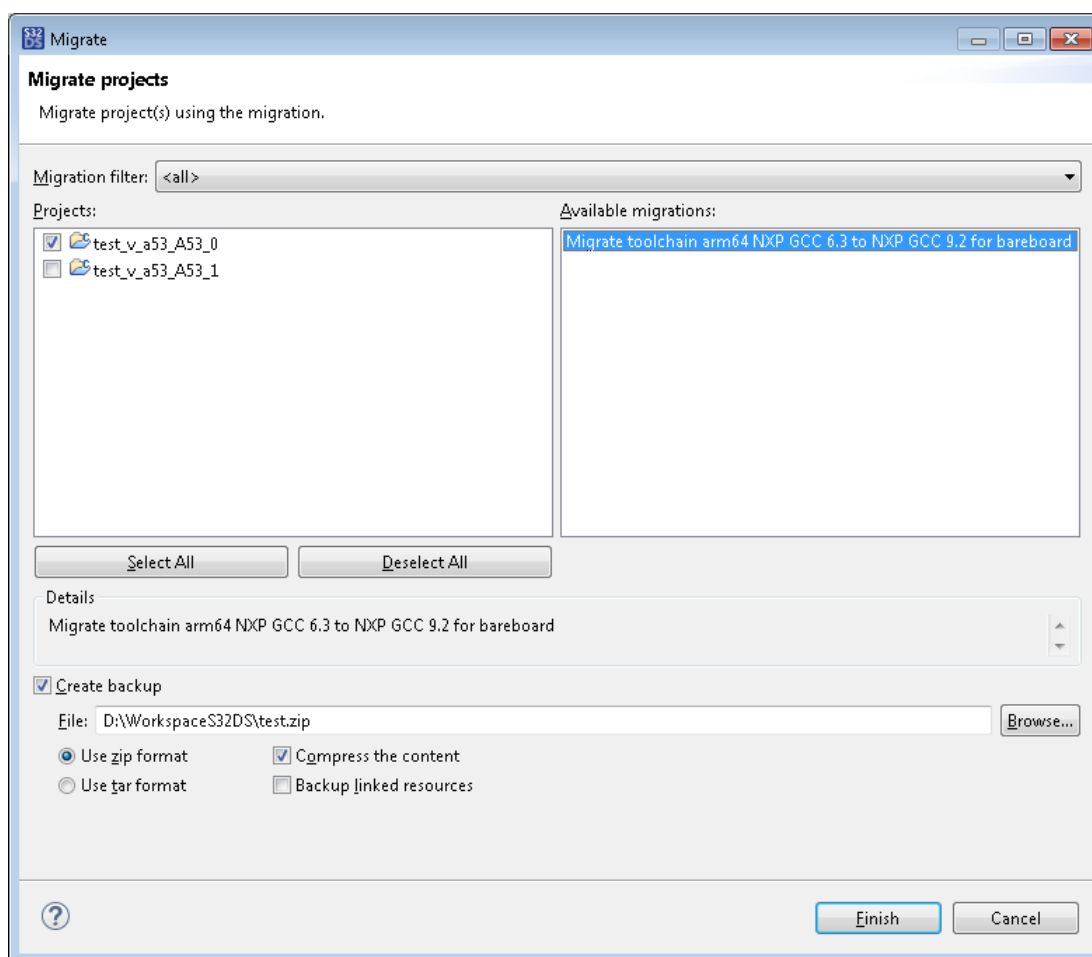
Note: Generally the SDK migration contains a toolchain migration in it.

Note: For toolchain migration refer steps 1-5 only.

The following explains how to migrate your project:

1. To open the Migrate wizard use one of the options:
 - Right click your project in the **Project Explorer** view and click **Migrate** on the context menu.
 - Click **File > Migrate**

The Migrate wizard appears.



2. Check that correct project is selected in the **Projects** tree.
3. Select migration type.
4. Check the **Create backup** checkbox and click **Browse** button to select the backup destination file.

Note: You can also select **Backup linked resources** and change the backup archive type.

5. Click **Finish** to confirm migration. System automatically creates new project with new SDK or toolchain and backups the original project.

Note: In case of conflicted files, all matched original files will be backed up and new project will contain the new ones.

Note: If some files were modified before the migration the **Save Resources** dialog will appear.

6. If the **Check conflicted files** dialog appears, click **OK** to confirm replacement of files with corresponding ones from new SDK.

The migration_output.txt file indicates the PE migrations fulfilled.

If the selected toolchain is not supported in the **New Project** wizard for the processor type of the original project, the migrated project may contain some errors.

If system generates any errors or warnings, you can see them in the **Problems** view. You can use the **Quick Fix** option if possible.

Troubleshooting

This section contains a series of tables that describe possible solutions to problems that may occur when using S32 Design Studio for S32 Platform. Each table contains:

- Symptoms that describe the sign or warning message for the type of problem.
- Possible solutions that describe what you should do to try to solve the problem.

The troubleshooting tables appear in the following order:

- [Licensing](#)
- [Installation](#)
- [Internet Access](#)
- [Project Files](#)
- [Building](#)
- [Debugging](#)

If your problem is not described below, check the list of known issues and workarounds in Release Notes, then refer to the S32 Design Studio for S32 Platform [community](#) or submit a [technical support](#) request.

For additional information about problems presumably relating to your device or included tools, refer to the following documentation:

Table 21: Related Documentation

Documentation for:	Location
S32 Debugger	Release Notes: /S32DS/tools/S32Debugger/Debugger/
S32 Configuration Tool	Release Notes: /Release_Notes/ S32 Configuration Tool Getting Started: Help > Help Contents
S32 Flash Tool	Release Notes and User Guide: /S32DS/tools/S32FlashTool/doc/
S32 Trace Tool	S32DS Software Analysis Documentation: Help > Help Contents , the pdf version is available in /S32DS/help/pdf/
S32 Debug Probe	User Guide: /S32DS/tools/S32Debugger/Debugger/Docs/
GNU Bare-Metal Targeted Tools for Arm 32-bit Embedded Processors	Release Notes: <ul style="list-style-type: none"> • GCC 6.3 - /S32DS/build_tools/gcc_b1620/gcc-6.3-arm32-eabi/ • GCC 9.2 - /S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi/ • GDB - /S32DS/tools/gdb_arm/arm32-eabi/
GNU Bare-Metal Targeted Tools for Arm 64-bit Embedded Processors	Release Notes: <ul style="list-style-type: none"> • GCC 6.3 - /S32DS/build_tools/gcc_b1620/gcc-6.3-arm64-eabi/ • GCC 9.2 - /S32DS/build_tools/gcc_v9.2/gcc-9.2-arm64-eabi/ • GDB - /S32DS/tools/gdb_arm/arm64-eabi/
GNU Linux Targeted Tools for Arm 64-bit Embedded Processors	Release Notes:

Documentation for:	Location
	<ul style="list-style-type: none"> • GCC 6.3 - /S32DS/build_tools/gcc_b1620/gcc-6.3-arm64-linux/ • GCC 9.2 - /S32DS/build_tools/gcc_v9.2/gcc-9.2-arm64-linux/ • GDB - /S32DS/tools/gdb_arm/arm64-linux/
SDK	Refer to the corresponding folder in /S32DS/software/
P&E GDB Server Plug-In	User Guide: /S32DS/help/pdf/
Hardware	Some software packages provide data sheets and reference manuals, refer to the corresponding folder in /S32DS/help/resources/manuals/ and /S32DS/help/resources/hardware/

Table 22: Licensing

Symptom	Possible solution
The product license cannot be activated	Ensure the Flexera server can be accessed from your workstation. Consult the network administrator of your company.

Table 23: Installation

Symptom	Possible solution
The "Insert New Media" message appears when installing S32DS 3.4	Make sure there is enough disk space for the product and temporary files. Free up disk space or click the Browse button to select a new location.
New packages and updates are not displayed in the S32DS Extensions and Updates tool	Toggle the New button in the S32DS Extensions and Updates window. This toggles off other filters that may block the new packages from being displayed.
	Check the Internet connection on your workstation. Reconnect if required.
	Ensure the path of the network repository is specified in the user preferences correctly: <ol style="list-style-type: none"> 1. Click Help > S32DS Extensions and Updates from the menu. 2. Click Manage Sites. 3. Ensure the list of available software sites includes the following location: http://www.nxp.com/lgfiles/updates/Eclipse/S32DS_3.4
	Ensure the network repository can be accessed from your workstation. Consult the network administrator of your company.
	Download the new packages and updates manually. Find the details in Downloading updates manually .

Table 24: Internet Access

Symptom	Possible solution
Wireless Internet permanently fails when debugging with S32 Debug Probe (USB) on a PC connected to Wi-Fi	Ask the network administrator to allow simultaneous domain and non-domain connections for the user account.
	Use a wired Ethernet connection to the domain network.
	Connect to an external wireless network, then connect to the domain through a VPN client.

Table 25: Project Files

Symptom	Possible solution
The source (header) files do not appear in the project's board folder after saving the device configuration to the project	Open the Pins, Clocks, or Peripherals perspective. Click Update Project > Open Update Project Dialog , select the files expected in the project's board folder, and click OK . Find the details in Editing a device configuration .

Table 26: Building

Symptom	Possible solution
The toolchain settings display the "Orphaned configuration" warning	The "Orphaned configuration. No base extension cfg exists..." message can be caused by missing toolchain. Make sure you installed the package that includes the tools necessary to build this type of project.
Build fails after updating the product version	The "Cannot run program: Launching failed. Error: Program not found in PATH" error can be caused by unresolved environment variables for the new product in old workspace. Create a new workspace and import the existing project.
Build fails with multiple unresolved symbols	The build failure can be caused by the incorrect C/C++ indexer settings. Indexing can be restricted by file size or by cache size, in which case the indexer may not update the database after some action was performed with a project file, for instance, because the file was too large. Find the details in Adjusting the C/C++ indexer settings for large files .
Build errors are reported without a particular location	The build failure can be caused by invalid characters in the resolved paths. Make sure that all your paths use the allowed characters only. Or, refer to Building projects in non-English versions of Windows .

Table 27: Debugging

Symptom	Possible solution
Debugging with Lauterbach TRACE32 fails in Linux	If the path to the debugged executable is too long, this may cause the TRACE32 debugger failure. Consider using a shorter path. As part of the solution, assign a shorter name to the project, recompile it, and start debugging anew.
Lauterbach TRACE32 running in Linux cannot find t32marm64, t32marm, t32mipu, t32mapex	Open the PROFILE file located in your home directory and add the following entry: <pre>export PATH="\$PATH:/opt/t32/bin/pc_linux64"</pre> where /opt/t32 is your Lauterbach installation directory. Modify the CM4 .cmm file to use the absolute paths to the reported executables (t32marm64, t32marm, and so on), for instance: <pre>os /opt/t32/bin/pc_linux64/t32marm64 -c ./Project_Settings/Debugger/CA53.t32</pre> <pre>os /opt/t32/bin/pc_linux64/t32mapex -c ./Project_Settings/Debugger/APU0.t32</pre>
Running a launch group results in unexpected termination of the secondary debug sessions	Open the launch group and increase the post-launch delay for the initial debug session. Find the details in Debugging on multiple cores .

Symptom	Possible solution
Debugging of multicore project fails	Previously created functional breakpoints (e.g. "main") are kept in the Breakpoints view. So the execution of <init> launch configuration tries to stop on it. All temporary functional breakpoints should be removed from the Breakpoints view (or disabled) before starting of a debug session.
Connecting to vpsession failed	The "com.synopsys.sls.core.CmdException: No session, operation cannot be completed" error can be caused by invalid environment variable value. If you have several Synopsys® Virtualizer Runtime versions, make sure the SNPS_VP_HOME value is set for the currently used version.

Part III

Reference

Topics:

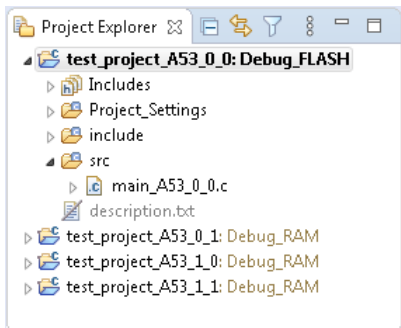
- [User interface](#)
- [Build configuration](#)
- [Folders and files](#)

User interface

Views and editors

Project Explorer view







The **Project Explorer** view displays a hierarchical view of all projects and their resources available in the current workspace.



The hierarchy includes the following levels (from top to bottom):

- Project names, each followed by the build configuration to be used.
On the image, the “Debug” build configuration will be applied to the project.
- Standard project folders (**Includes**, **src**, other).
Learn more about standard project folders in [Project folders and files](#).
- Nested folders, source files, and resources.
Note: Folders and files with crossed icons and faded font are excluded from the build.
- #include directives and definitions in code (see nested elements under **main.c** on the image)

The toolbar of the **Project Explorer** view includes the following buttons (from left to right):

	(Collapse All)	Click to collapse all nodes in the Project Explorer .
	(Link with Editor)	Click for the file currently opened in the editor to be highlighted in the Project Explorer .
	(Select and deselect filters to apply to the content in the tree)	Click to open the Filters and Customization dialog box.
	(View Menu)	Click to open the standard menu customizing the view.
	(Minimize)	Click to minimize the Project Explorer view. The Project Explorer icon appears at the left border of the main application window, next to the Restore icon.
	(Maximize)	Click to maximize the Project Explorer view. All other views are minimized and their icons appear at the right border of the main application window, each next to the dedicated Restore icon.

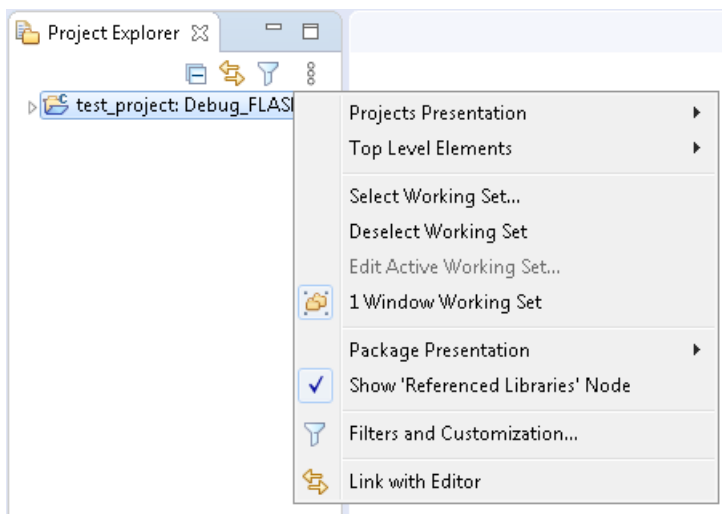
You can perform the following actions in the **Project Explorer** view:

- To expand or collapse a node in the hierarchy, double-click it.
- To collapse all nodes in the **Project Explorer**, click the **Collapse All** toolbar button located in the view.

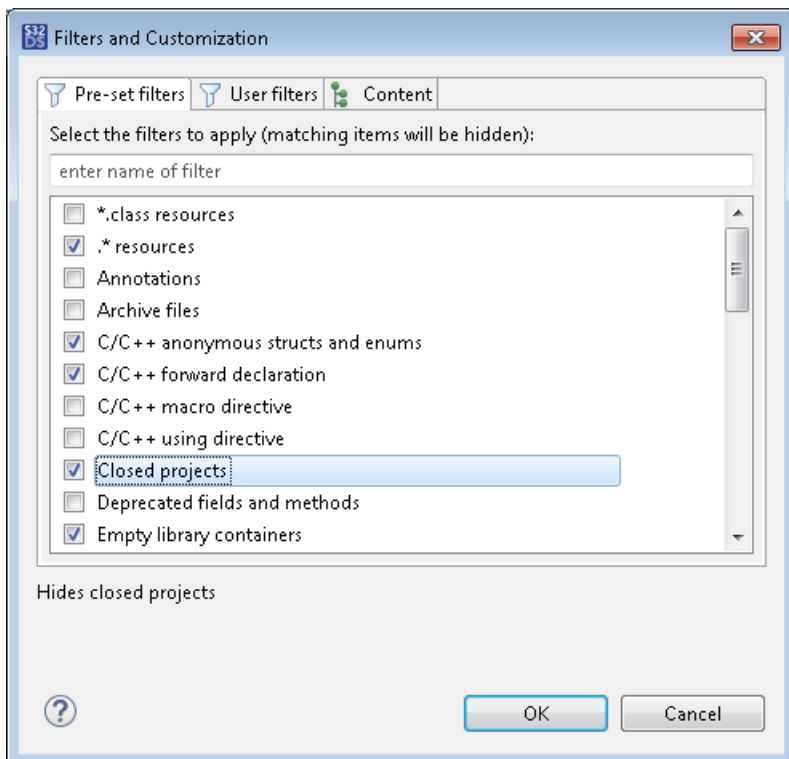
- To open a source file in the editor, double-click it in the **Project Explorer**, or drag and drop the file from the **Project Explorer** to the editor area.
- To close a project, select it in the **Project Explorer** and click **Project > Close Project** on the menu.
- To reopen a closed project, click the project name in the **Project Explorer** and use the **Project > Open Project** menu command.
- To permanently delete a project from the **Project Explorer** and workspace, and physically from the disc, use the **Edit > Delete** menu command.

To configure the **Project Explorer** to hide or display particular elements (closed projects, files types, definitions, and other):

1. Click the **View Menu** toolbar button and click **Customize View** on the context menu.



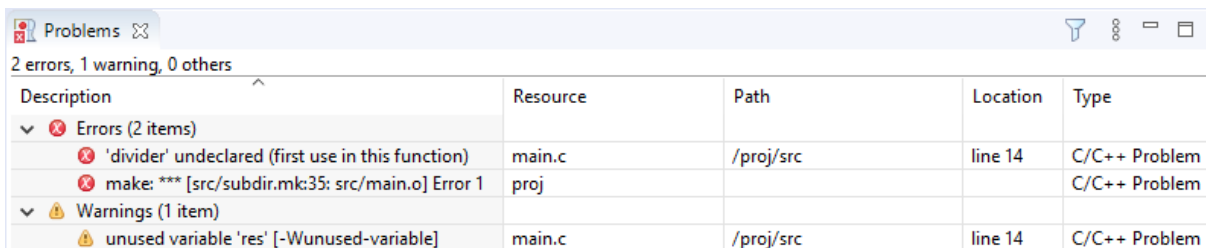
2. In the **Available Customizations** dialog box, go to the **Filters** tab. Select the options to be hidden.



- Click **OK**.

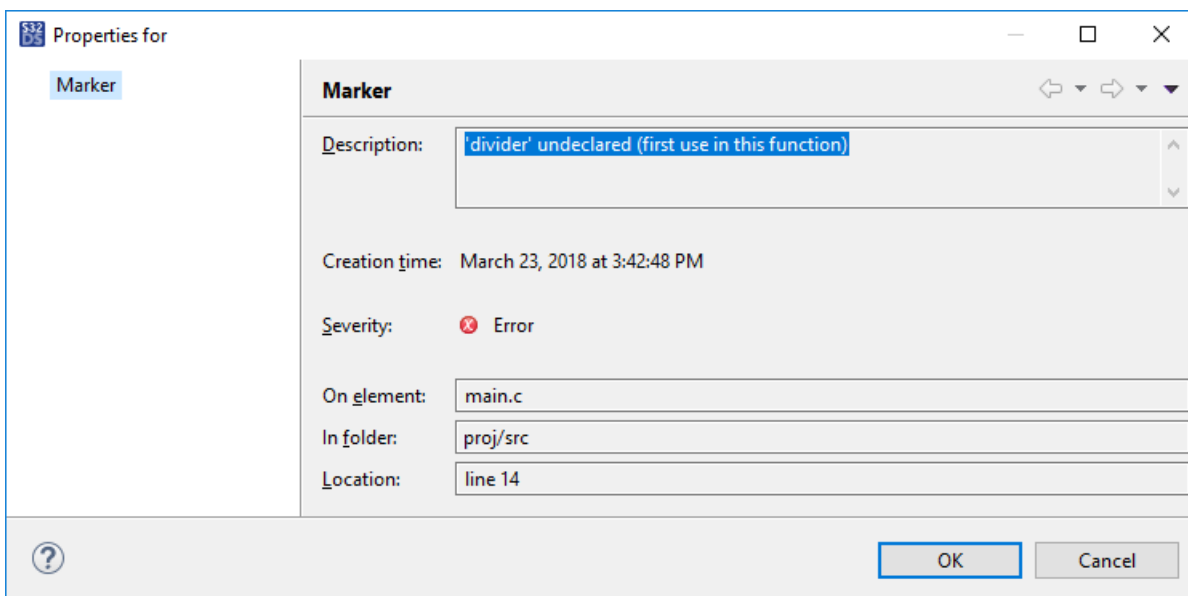
Problems view

The **Problems** view displays build errors, warnings, and information messages in a grid control. The detailed information about each issue includes a description, the resource, the path of the problem file, the location and the type of the issue. To jump to the location where a particular issue has been detected, double-click that issue in the grid.



Description	Resource	Path	Location	Type
Errors (2 items)				
'divider' undeclared (first use in this function)	main.c	/proj/src	line 14	C/C++ Problem
make: *** [src/subdir.mk:35: src/main.o] Error 1	proj			C/C++ Problem
Warnings (1 item)				
unused variable 'res' [-Wunused-variable]	main.c	/proj/src	line 14	C/C++ Problem

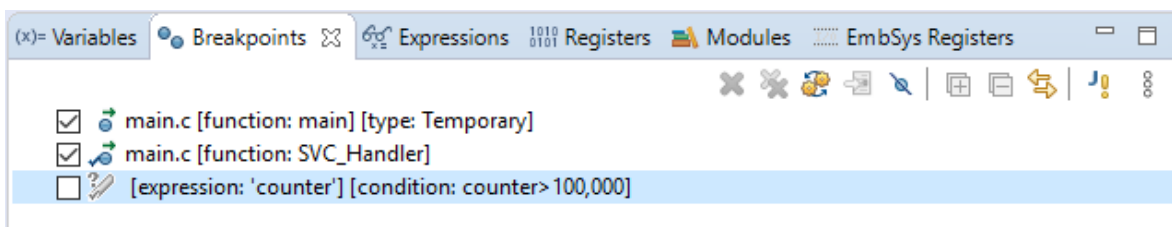
Right-click an issue in the grid and click **Properties** on the context menu. The **Properties** dialog box appears to display the detailed description of the issue:



Breakpoints view

The **Breakpoints** view lists all the breakpoints set in the workbench projects. This view also allows breakpoints to be grouped by type, project, file, or working sets, and supports nested groupings. If you double-click a breakpoint displayed by this view, the source code editor displays the source code statement on which this breakpoint is set.

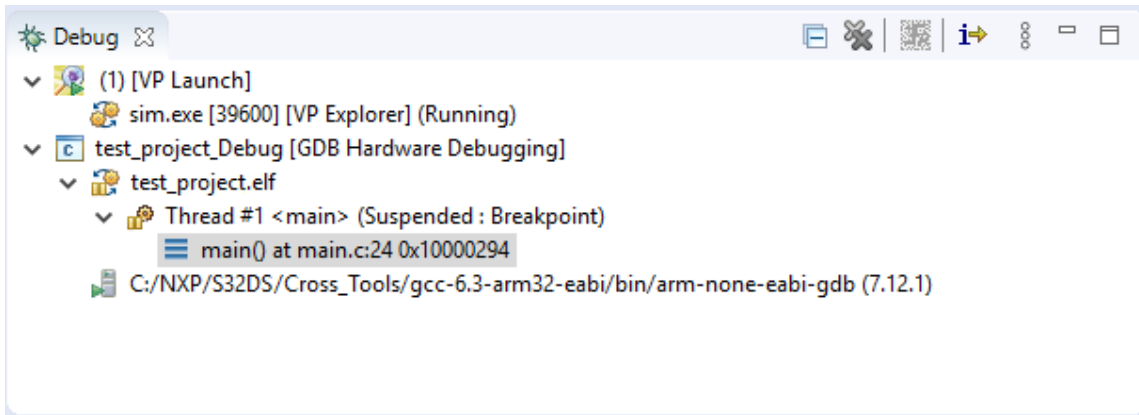
To open the **Breakpoints** view, click **Window > Show View > Breakpoints** on the menu. Or, click **Window > Show View > Other** and find **Breakpoints** in the **Show View** dialog box.



Debug view

The **Debug** view shows the information about current debug sessions in a tree hierarchy.

To display the **Debug** view, click **Window > Show View > Other... > Debug > Debug** on the menu:



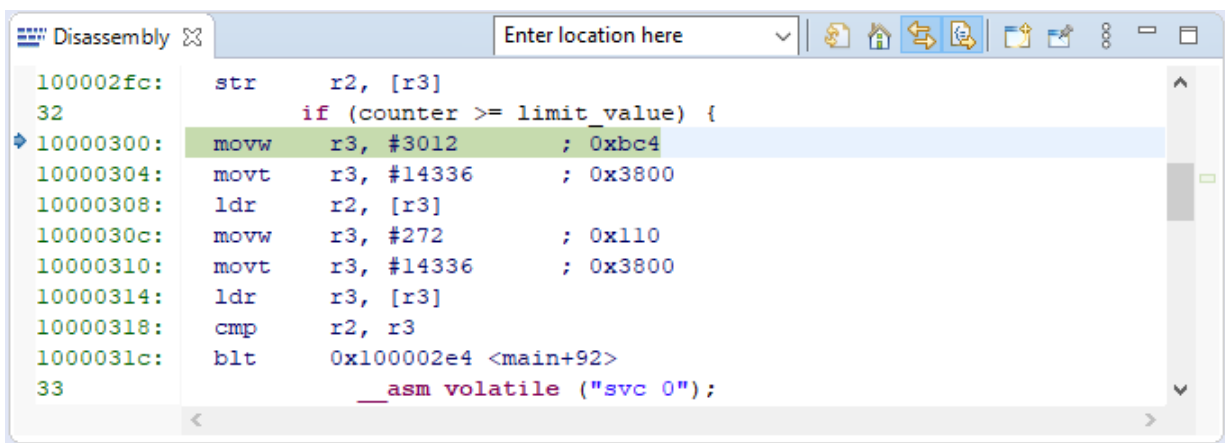
Use the **Debug** view to perform the following tasks:

- Clear all terminated processes
- Start a new debug session for the selected process
- Resume execution of the currently suspended debug target
- Halt execution of the currently selected thread in a debug target
- Terminate the selected debug session and/or process
- Detach the debugger from the selected process
- Execute the current line, including any routines, and proceed to the next statement
- Execute the current line, following execution inside a routine
- Re-enter the selected stack frame
- Examine a program as it steps into disassembled code

Disassembly view

The **Disassembly** view shows the loaded program as assembly language instructions mixed with source code for comparison. The next instruction to be executed is indicated by an arrow marker and highlighted in the view.

To display the **Disassembly** view, click **Window > Show View > Other... > Debug > Disassembly** on the menu.



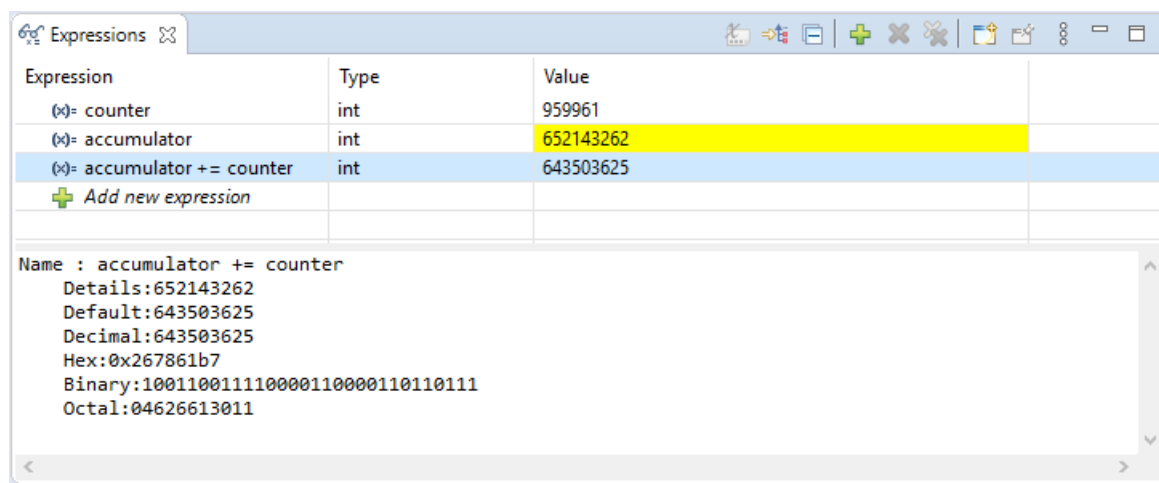
You can perform the following tasks in the **Disassembly** view:

- Set breakpoints at the start of any assembly language instruction
- Enable and disable breakpoints and set their properties
- Step through the disassembled instructions of your program
- Jump to specific instructions in the program

Expressions view



The **Expressions** view helps you inspect data from a stack frame of a suspended thread. In contrast to the **Variables** view that shows variables in the current scope, the **Expressions** view can monitor the values of static and global variables and executed statements that you add to the view.

To open the **Expressions** view in the current perspective, click **Window > Show View > Other > Debug > Variables** on the menu:




To open additional **Expressions** views, click  (Open New View) on the toolbar above the view.

To add an expression to the **Expressions** view, do any of the following:

- Copy an expression (a variable name or a statement) from the file opened in the editor area, click  **Add new expression** in the grid, and paste the expression to the new grid line.
- Select an expression in the opened file, right-click and click **Add Watch Expression** on the context menu.
- Click  (Create a new watch expression) on the toolbar above the view and enter an expression in the **Add Watch Expression** dialog box.

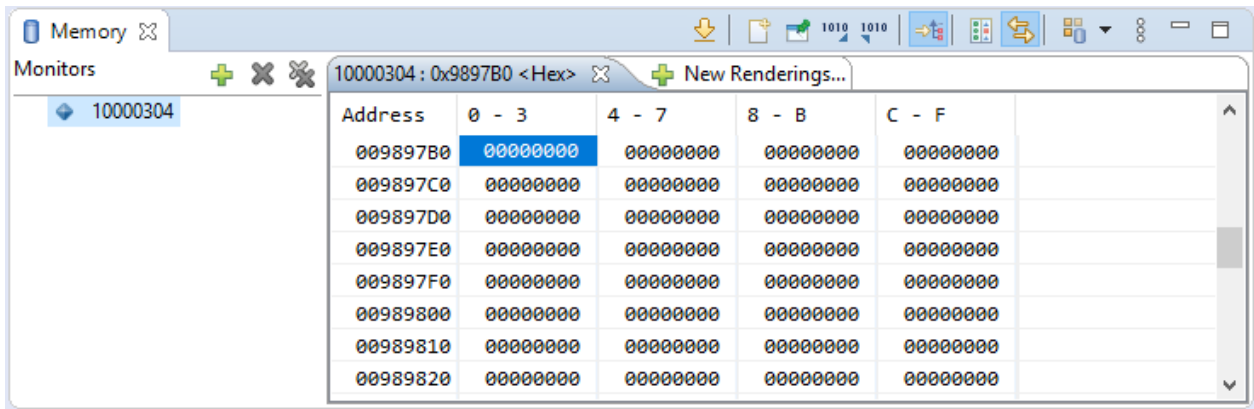
When added to the **Expressions** view at debug time, expressions are displayed in the grid with their data types and actual values that are updated in real time. Click an expression in the grid to view more detailed information about it in the **Detail** pane below the grid.

When the debug session is terminated, the expression names remain until deleted manually. To delete all expressions, click  (Remove All Expressions) on the toolbar above the view.

Memory view

The **Memory** view allows you to monitor and modify your process memory. The process memory is presented as a list of memory monitors. Each monitor represents a section of memory specified by its location called base address. Each memory monitor can be displayed in one of the predefined data formats.

To open the **Memory** view, click **Window > Show View > Other... > Debug > Memory** on the menu:



By default, the **Memory** view displays the layout and formats that were set in the previous debug session.

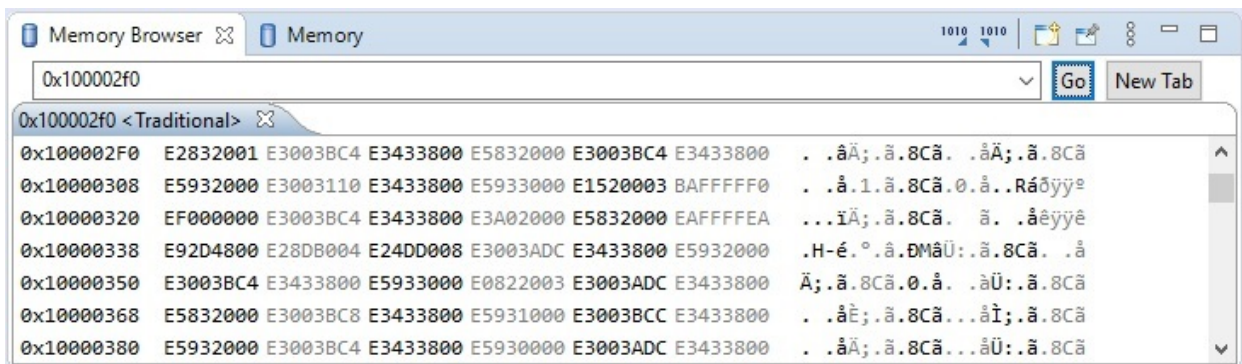
- Add your memory monitors in the **Monitors** pane. The default rendering pane without addresses is displayed automatically. The addresses become visible after the first step in the debugger.
- Add more rendering panes for a monitor. For each rendering pane, specify the rendering type from the list:
 - Floating Point
 - Traditional
 - Hex
 - ASCII
 - Signed Integer
 - Unsigned Integer
 - Hex Integer

Warning: Expressions with the unary increment, decrement and assignment operators used in the **Memory** and **Expressions** views modify memory and may cause side effects.

Memory Browser view

The **Memory Browser** view serves for monitoring particular locations in process memory.

To display the **Memory Browser** view, click **Window > Show View > Other... > Debug > Memory Browser** on the menu.



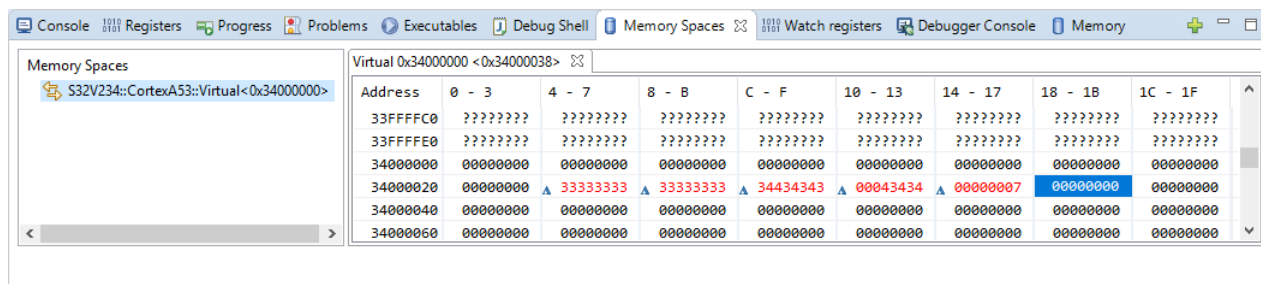
To browse to a desired memory location, type the memory address in the memory address box and click **Go**. The memory location is displayed in the tab of the **Memory Browser** view. You can add more tabs by clicking **New Tab**.

Memory Spaces view

The **Memory Spaces** view enables access to the selected memory spaces on the target device.

Note: Memory spaces are available only during a debug session supporting memory space operating by the **S32 Debugger**.

The **Memory Spaces** view is included in the **Debug** perspective by default. If the view was closed, reopen it from the menu by selecting **Window > Show View > Other > Debug > Memory Spaces**.







The memory spaces are presented as a list of memory spaces in the left part of the view. Each space represents a section of memory specified by its location called address.

The color indicates the state:


- Gray - initial state,
- Black - data was read,
- Red - data changed since first-time read.

The "?" mark indicates that memory can't be read (write-only, restricted, etc.).

The toolbar of the **Memory Spaces** view includes the following buttons (from left to right):

	(Add memory space)	Click to add a memory space to the list in the Memory Spaces view.
	(Minimize)	Click to minimize the Memory Spaces view.
	(Maximize)	Click to maximize the Memory Spaces view. All other views are minimized and their icons appear at the right border of the main application window, each next to the dedicated Restore icon.
	(Restore)	Click to restore the Memory Spaces view from minimal or maximal state.

During debugging you can perform the following actions in the **Memory Spaces** view:

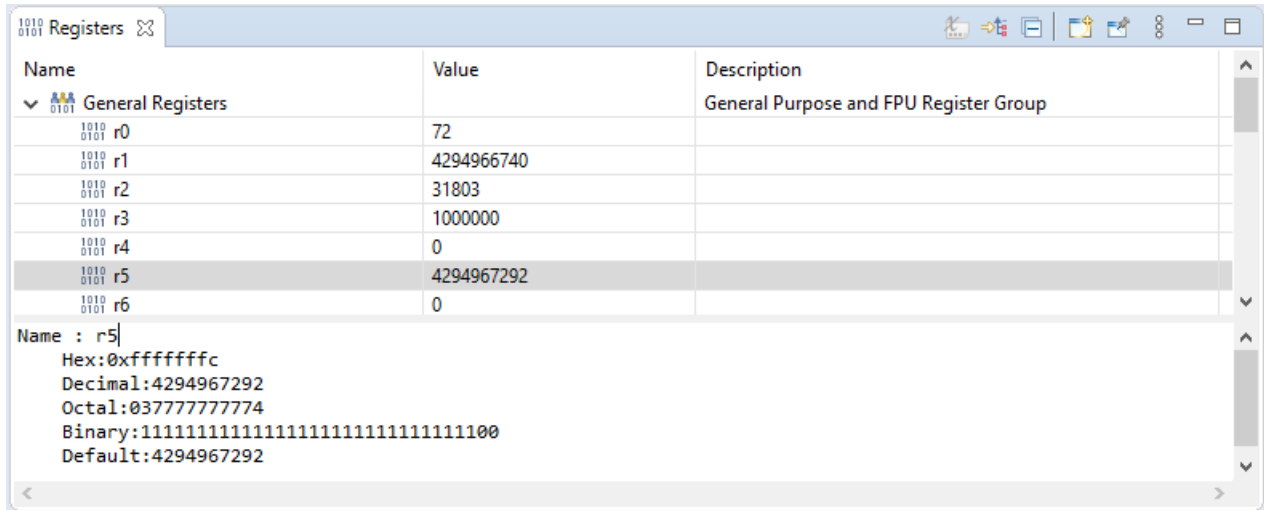
- To add new memory space, click the  button on the toolbar or right-click **Memory Spaces** column and select **Add** from context menu. This action opens the **Add memory space** dialog. Type the memory address in the **Address (HEX)** field (HEX value can be set both with or without "0x" prefix), select available **Memory space** from the drop-down menu and click **Select**. The memory space will be added to the list in the left pane.
- To write data, double-click the cell in the view, type in data and press **Enter** or continue typing - when the available amount of symbols is reached, the system will automatically write the data and move to the next cell.
- To delete a memory space, select the space in the list, right-click on it and select **Remove** from context menu.

The **Memory Spaces** view remembers the list of added spaces for each running launch configuration (until the list is cleared or debug session is terminated).

Registers view

The **Registers** view lists information about the registers in a selected stack frame. Values that have changed are highlighted in the **Registers** view when your program stops. You can use the **Registers** view to look into register details and change register values.

To display the **Registers** view, switch to the **Debug** perspective and click **Window > Show View > Other... > Debug > Registers** on the menu.



You can change the positional numeral system in which the debugger displays register values. The following numeral systems are supported:

- Default
- Decimal
- Hexadecimal
- Octal
- Binary

Note: Casting a register to a type requires the size of the register to match the size of the type, otherwise the cast will fail. Therefore, if the type is a complex one (for example, structure, union), it should be declared first to avoid padding done by compilers.

EmbSys Registers view

The **Embedded Systems Registers** view enables access to the peripheral registers of the target hardware.

- When a debug session is on, the **EmbSys Registers** view can read and display the actual values of the selected registers. The displayed register values can be exported and imported to/from a text file.
- When not in a debug session, the **EmbSys Registers** panel provides the structured view of the peripheral registers on the target device.

Note: S32 Debugger implements different mechanisms for **Peripheral Registers** and **EmbSys Registers** views when accessing SoC memory mapped registers. **Peripheral Registers view** is a preferred option since S32 Debugger is accessing these registers directly via the system bus, bypassing caches and MMU (if present in the core being debugged).

The **EmbSys Registers** view is included in the **Debug** perspective by default. If the view was closed, reopen it from the menu by selecting **Window > Show View > Other > Debug > EmbSys Registers**.

Register	Hex	Bin	Reset	Access	Address	Description
DMA						Enhanced direct memory access co
DMA						Enhanced direct memory access co
16011011 CR	- not read -		0x00000400	RW	0x40002000	Control Register
16011011 ES	- not read -		0x00000000	RO	0x40002004	Error Status Register
16011011 ERQ	- not read -		0x00000000	RW	0x4000200C	Enable Request Register
16011011 EEI	- not read -		0x00000000	RW	0x40002014	Enable Error Interrupt Register
10101011 CEEI	- write onl...		0x00	WO	0x40002018	Clear Enable Error Interrupt Register
10101011 SEEI	- write onl...		0x00	WO	0x40002019	Set Enable Error Interrupt Register

The **EmbSys Registers** view presents the information about the peripheral registers in a tabular format. The **Register** field displays the peripheral registers arranged in a hierarchy with the following levels (from top to bottom): a category, a group of registers, a register, a bit (bit number). Any node in the hierarchy can be expanded or collapsed with a double click.

The **Description** field provides the information about the object shown in the **Register** field.

The remaining fields are populated for the “register” nodes:



- **Hex:** Displays the value read from the register in the hexadecimal format.
- **Bin:** Displays the same register value in the binary format.
- **Reset:** Displays the reset value of the register in the hexadecimal format, for instance, 0x00000000. A mouse cursor moved over the hex value displays a tooltip with the binary equivalent.
Note: A “zero” value is displayed if the reset value is either set to 0x00000000 or not defined for the register.
- **Access:** Displays access to the register (Read-only, Read-Write, Write-only).
- **Address:** Displays the register address in memory.

Some registers are displayed in the **Register** field with the “+” sign preceding the register name. These register names are aliases of one register showing the same address in the **Address** field:

Register	Hex	Bin	Reset	Access	Address	Description
SPI						Serial Peripheral Interface
SPI_0						Serial Peripheral Interface
10101011 MCR			0x00004001	RW	0x40057000	Module Configuration Register
10101011 TCR			0x00000000	RW	0x40057008	Transfer Count Register
10101011 + CTAR0			0x78000000	RW	0x4005700C	Clock and Transfer Attributes Register (In Master Mod
10101011 + CTAR_SLAVE			0x78000000	RW	0x4005700C	Clock and Transfer Attributes Register (In Slave Mode)

The toolbar of the **EmbSys Registers** view includes the following buttons (from left to right):

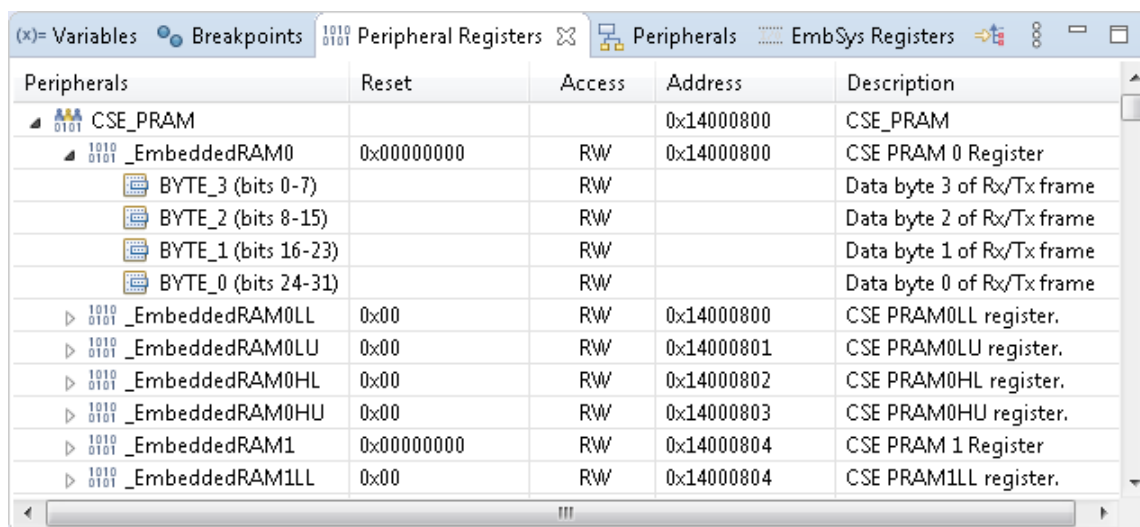
	(EmbSysRegView Project Properties)	Click to open the configuration settings of the EmbSys Registers view in the properties of a project selected in the Project Explorer .
	(Copy selection to clipboard)	Click to copy the Register , Hex , and Address fields from the selected row(s) to the clipboard.
	(Export selection to file)	Click to export values of the selected or all peripheral registers to an XML file. Learn more in Exporting peripheral register values .
	(Import from file)	Click to import values from an XML file and write them to peripheral registers. Learn more in Importing peripheral register values .
	(Collapse All)	Click to collapse all nodes in the EmbSys Registers view.

	(Minimize)	Click to minimize the EmbSys Registers view.
	(Maximize)	Click to maximize the EmbSys Registers view. All other views are minimized and their icons appear at the right border of the main application window, each next to the dedicated Restore icon.

Peripheral Registers view

The **Peripheral Registers** view provides the structured view of the peripheral registers on the target hardware. If Arm® core registers are defined for the target device they are also present in this view.

The **Peripheral Registers** view is included in the **Debug** perspective by default. If the view was closed, reopen it from the menu by selecting **Window > Show View > Other > Debug > Peripheral Registers**.






The **Peripheral Registers** view presents the information about the peripheral registers in a tabular format. The **Peripherals** column displays the registers arranged in a hierarchy with the following levels (from top to bottom): a group of registers, a peripheral, a cluster (if set), a register, a field (sequential bits within a register). Any node in the hierarchy can be expanded or collapsed with a click.



The **Description** column provides the information about the object shown in the **Peripherals** column.

The remaining columns are populated for the “register” nodes:

- **Reset:** Displays the reset value of the register in the hexadecimal format, for instance, 0x00000000. A mouse cursor moved over the hex value displays a tooltip with the binary equivalent.
- **Note:** A “zero” value is displayed if the reset value is either set to 0x00000000 or not defined for the register.
- **Access:** Displays the register access type (Read-Only, Read-Write, Write-Only).
- **Address:** Displays the register address in memory.

The toolbar of the **Peripheral Registers** view includes the following buttons (from left to right):

	(Show structure)	Toggle for the Peripheral Registers view to show peripheral registers combined in categories.
	(View Menu)	Click to open the menu customizing the layout of the Peripheral Registers view. Options: <ul style="list-style-type: none"> • Restore table layout: Sets the layout to its default.
	(Minimize)	Click to minimize the Peripheral Registers view.

 (Maximize)	Click to maximize the Peripheral Registers view. All other views are minimized and their icons appear at the right border of the main application window, each next to the dedicated Restore icon.
 (Restore)	Click to restore the Peripheral Registers view from minimal or maximal state.

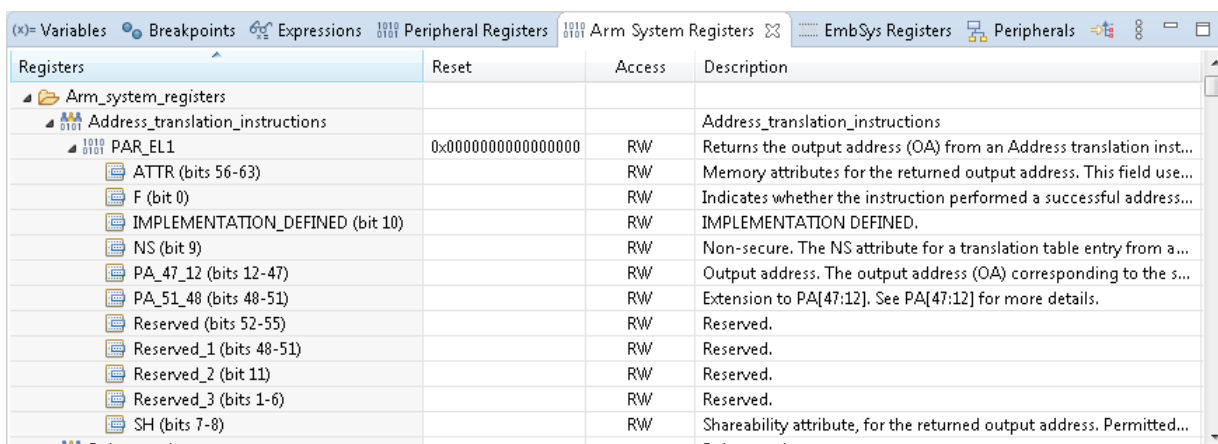
The **Peripheral Registers** view context menu has the following options:

- **Find**: Click to open the **Find Element** dialog. Search is available by register name, description or address.
- **Copy**: Click to copy the selected registers text info to a system clipboard.
- **Watch Register(s)**: Click to send the selected registers to appear in the **Watch registers view**.

Arm System Registers view

The **Arm System Registers** view provides the structured view of the Arm System registers (if available) of the target hardware.

The **Arm System Registers** view is included in the **Debug** perspective by default. If the view was closed, reopen it from the menu by selecting **Window > Show View > Other > Debug > Arm System Registers**.



Registers	Reset	Access	Description
Arm_system_registers			
Address_translation_instructions			Address_translation_instructions
PAR_EL1	0x0000000000000000	RW	Returns the output address (OA) from an Address translation inst...
ATTR (bits 56-63)		RW	Memory attributes for the returned output address. This field use...
F (bit 0)		RW	Indicates whether the instruction performed a successful address...
IMPLEMENTATION_DEFINED (bit 10)		RW	IMPLEMENTATION DEFINED.
NS (bit 9)		RW	Non-secure. The NS attribute for a translation table entry from a...
PA_47_12 (bits 12-47)		RW	Output address. The output address (OA) corresponding to the s...
PA_51_48 (bits 48-51)		RW	Extension to PA[47:12]. See PA[47:12] for more details.
Reserved (bits 52-55)		RW	Reserved.
Reserved_1 (bits 48-51)		RW	Reserved.
Reserved_2 (bit 11)		RW	Reserved.
Reserved_3 (bits 1-6)		RW	Reserved.
SH (bits 7-8)		RW	Shareability attribute, for the returned output address. Permitted...



The **Arm System Registers** view presents the information about the Arm System registers in a tabular format. The **Peripherals** column displays the registers arranged in a hierarchy with the following levels (from top to bottom): a group of registers, a peripheral, a cluster (if set), a register, a field (sequential bits within a register). Any node in the hierarchy can be expanded or collapsed with a click.




The **Description** column provides the information about the object shown in the **Registers** column.

The remaining columns are populated for the “register” nodes:

- **Reset**: Displays the reset value of the register in the hexadecimal format, for instance, 0x00000000. A mouse cursor moved over the hex value displays a tooltip with the binary equivalent.
- **Note**: A “zero” value is displayed if the reset value is either set to 0x00000000 or not defined for the register.
- **Access**: Displays the register access type (Read-Only, Read-Write, Write-Only).

The toolbar of the **Arm System Registers** view includes the following buttons (from left to right):

 (Show structure)	Toggle for the Arm System Registers view to show registers combined in categories.
 (View Menu)	Click to open the menu customizing the layout of the Arm System Registers view. Options: <ul style="list-style-type: none"> • Restore table layout: Sets the layout to its default.

	(Minimize)	Click to minimize the Arm System Registers view.
	(Maximize)	Click to maximize the Arm System Registers view. All other views are minimized and their icons appear at the right border of the main application window, each next to the dedicated Restore icon.
	(Restore)	Click to restore the Arm System Registers view from minimal or maximal state.

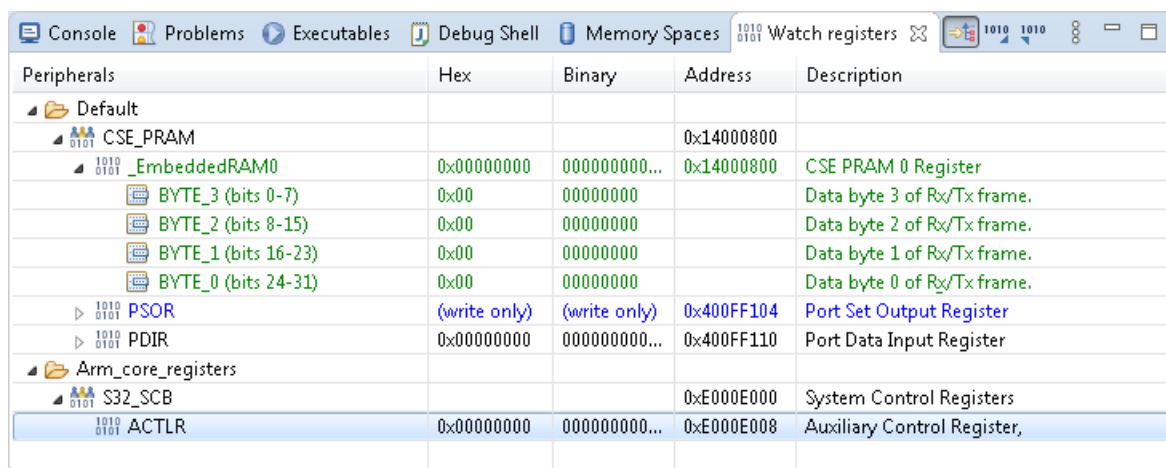
The **Arm System Registers** view context menu has the following options:

- **Find:** Click to open the **Find Element** dialog. Search is available by register name or description.
- **Copy:** Click to copy the selected registers text info to a system clipboard.
- **Watch Register(s):** Click to send the selected registers to appear in the [Watch registers view](#).

Watch registers view

The **Watch registers** view enables access to the selected registers on the target device added from the [Peripheral Registers view](#) or the [Arm System Registers view](#).

The **Watch registers** view is included in the **Debug** perspective by default. If the view was closed, reopen it from the menu by selecting **Window > Show View > Other > Debug > Watch registers**.



Peripherals	Hex	Binary	Address	Description
Default				
CSE_PRAM			0x14000800	
EmbeddedRAM0	0x00000000	000000000...	0x14000800	CSE PRAM 0 Register
BYTE_3 (bits 0-7)	0x00	00000000		Data byte 3 of Rx/Tx frame.
BYTE_2 (bits 8-15)	0x00	00000000		Data byte 2 of Rx/Tx frame.
BYTE_1 (bits 16-23)	0x00	00000000		Data byte 1 of Rx/Tx frame.
BYTE_0 (bits 24-31)	0x00	00000000		Data byte 0 of Rx/Tx frame.
PSOR	(write only)	(write only)	0x400FF104	Port Set Output Register
PDIR	0x00000000	000000000...	0x400FF110	Port Data Input Register
Arm_core_registers				
S32_SCB			0xE000E000	System Control Registers
ACTLR	0x00000000	000000000...	0xE000E008	Auxiliary Control Register,

The **Watch registers** view presents the information about the selected registers in a tabular format. The **Peripherals** column displays the registers arranged in a hierarchy with the following levels (from top to bottom): a group of registers, a peripheral, a cluster (if set), a register, a field (sequential bits within a register). A minimal unit to be watched is a register. Any node in the hierarchy can be expanded or collapsed with a double click.

The **Description** column provides the information about the object shown in the **Peripherals** column.

The remaining columns are populated for the “register” nodes:

- **Hex:** Displays the value read from the register in the hexadecimal format. If the register is not readable the field shows the access info (write-only).
- **Binary:** Displays the same register value in the binary format. If the register is not readable the field shows the access info (write-only).
- **Address:** Displays the register address in memory.

Note: For Arm System registers the field is empty.








The color indicates the access type:

- Black - Read-Only
- Green - Read-Write,

- Blue - Write-Only.

The italics font indicates the **Read on demand** state of a register.

The toolbar of the **Watch registers** view includes the following buttons (from left to right):

	(Show structure)	Toggle for the Watch registers view to show peripheral registers combined in categories.
	(Import registers from file)	Click to import registers from an XML file and write them to target registers. Learn more in Importing registers values .
	(Export registers to file)	Click to export values of the selected registers to an XML file. Learn more in Exporting registers values .
	(View Menu)	Click to open the menu customizing the layout of the Watch registers view. Options: <ul style="list-style-type: none"> • Restore table layout: Sets the layout to its default. • Show Full Path: Displays registers in path mode (group/register).
	(Minimize)	Click to minimize the Watch registers view.
	(Maximize)	Click to maximize the Watch registers view. All other views are minimized and their icons appear at the right border of the main application window, each next to the dedicated Restore icon.
	(Restore)	Click to restore the Watch registers view from minimal or maximal state.

The **Watch registers** view context menu has the following options:

- **Find**: Click to open the **Find Element** dialog. Search is available by register name, description or address (if available for a register).
- **Copy**: Click to copy the selected registers text info to a system clipboard.
- **Remove Register(s)**: Click to remove the selected registers from the **Watch registers** view.
- **Import**: Click to import registers from an XML file.
- **Export**: Click to export the selected registers values to an XML file.
- **Read always**: Click to enable reading of the selected registers.

Note: Registers with side effects (readAction) are always kept in the **Read on demand** state and can't be reset to the **Read always**.

- **Read on demand**: Click to mark the selected registers be read only on demand.
- **Read**: Click to read values of the selected registers.

During debugging the **Watch registers** view can read, display and write the values of the registers.

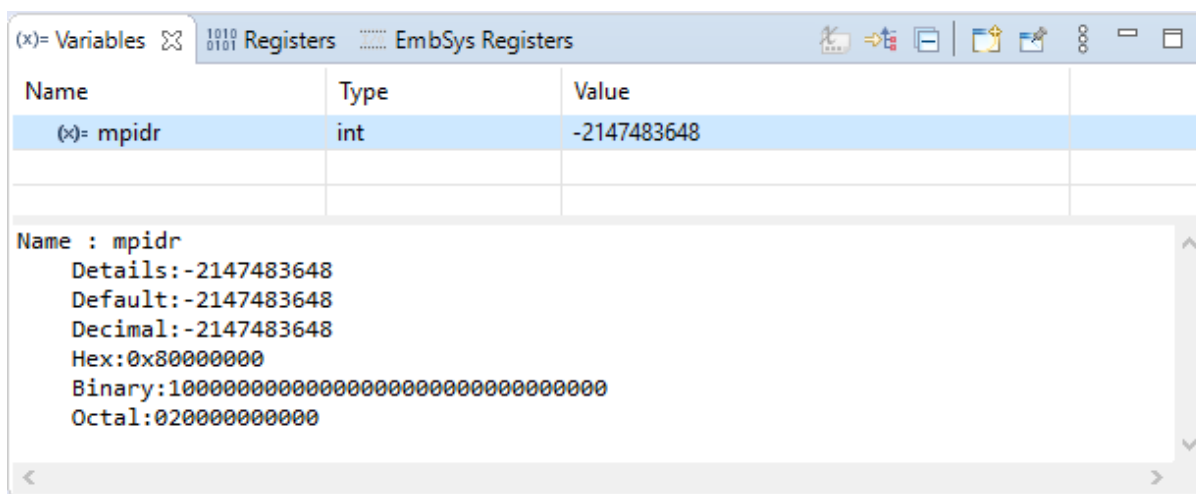
Switching between the nodes in the **Debug** view also results in the loss of context in the **Watch registers** view. If you switch from the thread of a running program to a different node such as the GDB client or other, the **Watch registers** view stops reading the registers from the connected device. To restore the debug context, click the **main()** node of the project.

The **Watch registers** view remembers the state and apply it to any other launch configuration with the same SVD source.

Variables view

The **Variables** view shows all static variables for each process that you debug (global variables are displayed in the **Expression** view). Use the view to observe changes in variable values as the program executes in the currently selected stack frame.

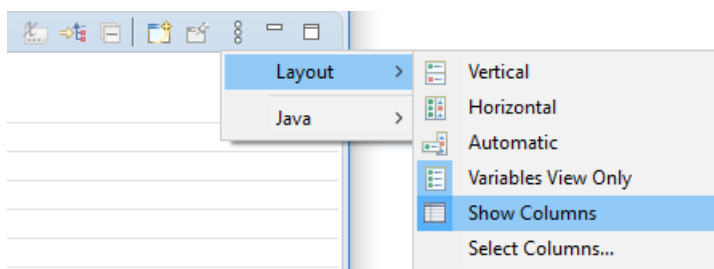
To open the **Variables** view, click **Window > Show View > Other... > Debug > Variables** on the menu:



The toolbar of the **Variables** view includes the following buttons (from left to right):

	(Show Type Names)	Toggle for the Variables view to show type names when the view is configured not to show columns.
	(Show Logical Structure)	Toggle for the Variables view to show the logical structure of variables.
	(Collapse All)	Click to collapse all nodes in the Variables view.
	(Open New View)	Click to open one more Variables view next to the existing one.
	(Pin to Debug Context)	Click to pin the Variables view next to the Debug view.
	(View Menu)	Click to open the menu customizing the layout of the Variables view.
	(Minimize)	Click to minimize the Variables view.
	(Maximize)	Click to maximize the Variables view.

To configure the **Variables** view, click the **View Menu** toolbar button. Use the **Layout** menu commands to configure the look of the **Variables** view:



- **Vertical:** Click to display the detail pane at the bottom of the view, aligning the parts of the view vertically. The detail pane displays the detailed information about a selection.
- **Horizontal:** Click to display the detail pane at the right side of the view, aligning the parts of the view horizontally.
- **Automatic:** Click for the view to set the layout automatically.

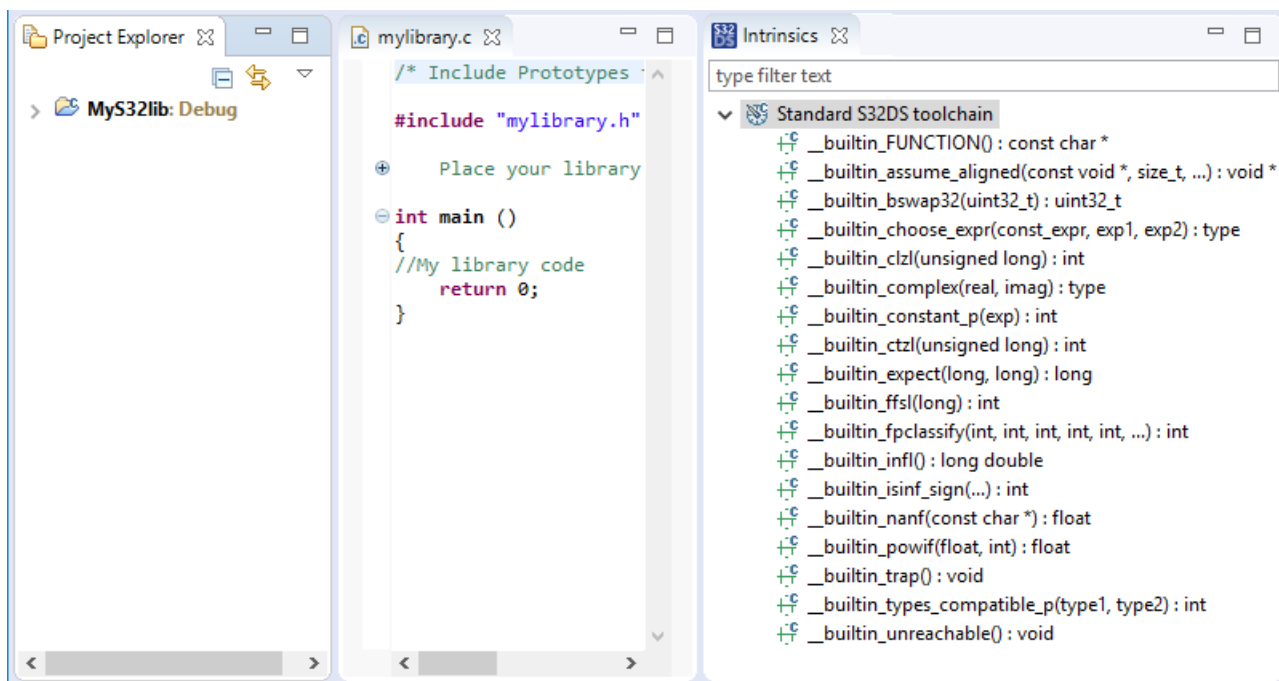
- **Variables View Only:** Click to hide the detail pane.
- **Show Columns:** Toggle to show columns in the view.
- **Select Columns:** Click to open the **Select Columns** dialog box and select the columns to be displayed. This option appears on the menu if the **Show Columns** option is toggled.

Intrinsics view

The **Intrinsics** view displays the GCC functions that are built into the S32 Design Studio compiler. These built-in functions are referred to as intrinsic functions or intrinsics. If a C/C++ project includes the Standard S32DS toolchain, the supported intrinsics can be used in that project's code for optimization.

To open the **Intrinsics** view, click **Window > Show View > Other** on the menu. In the **Show View** dialog box, expand the **Other** section and select **Intrinsics**. Click **OK**.

For the **Intrinsics** view to display data, click any project file or folder in the **Project Explorer** view. If the intrinsics are supported in the project, the **Intrinsics** view displays the list of built-in functions.



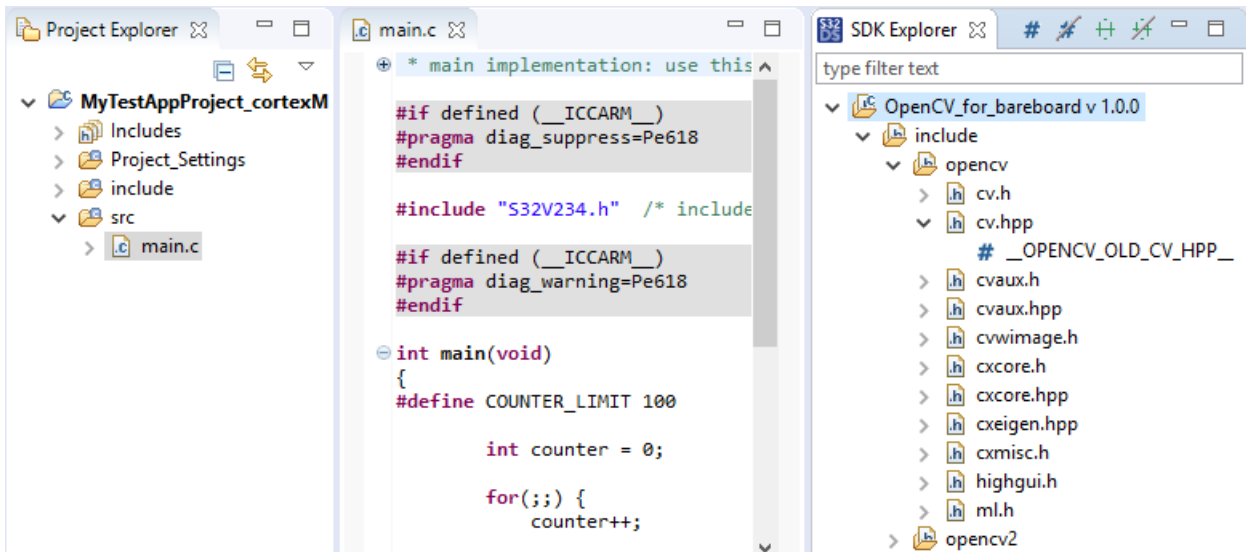
To add an intrinsic to your code, drag and drop it from the **Intrinsics** view to a proper place in the opened source file. This action adds the function call to the selected place in the file.

SDK Explorer view

The **SDK Explorer** view displays the structure of SDKs belonging to the active project.

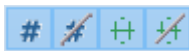
To open the **SDK Explorer** view, click **Window > Show View > Other**. In the **Show View** dialog box, expand the **Other** section and select **SDK Explorer**. Click **OK**.

Select any project file or folder in **Project Explorer**. The **SDK Explorer** displays the include folder for each SDK attached to the project, with nested folders and header files inside:



You can perform the following actions in the **SDK Explorer** view:

- To expand or collapse a folder, click the "arrow" button near the folder.
- To see all definitions and declarations of a header file (such as macros, functions, namespaces, enumerations, and other), click the "arrow" button near the file. If a file does not display the "arrow" button, it does not include any definitions and declarations.
- To see all files, definitions, and declarations that include a particular string pattern, type the string in the filter box.
- To open a header file in the file editor, double-click it.
- To see a particular definition or declaration in the code, double-click it in the view. This action opens the header file in the file editor and highlights the respective line.
- To hide from the view active macros, inactive macros, active functions, and inactive functions, toggle the respective buttons on the top:



Note: Inactive macros and functions are those located inside the `#if defined` and `#endif` constructs intended for a different type of a compiler. These sections are grayed out in the file editor.

- To add an SDK function to your code, drag and drop it from the **SDK Explorer** view to a proper place in the opened source file. This action inserts the `#include` statement for the corresponding header file and adds the function call to the selected place in the file.

Editor area

The editor area can be used to open text editors associated with different types of files. When you double-click a source file in the **Project Explorer** view, the associated editor opens the selected file. The following elements indicate modifications that took place in the file:

```

main.c
+ #define read_mpidr() ({
    #define clusterid(mpidr) ((mpidr >> 8) & 1)
    #define coreid(mpidr) (mpidr & 1)

    int clusterid, coreid, counter, accumulator = 0, limit_value = 1000000;

- int main(void) {
    int mpidr = read_mpidr();
    clusterid = clusterid(mpidr);
    coreid = coreid(mpidr);
    counter = 0;
    for (;;) {
        counter++;

        if (counter >= limit_value) {
            __asm volatile ("svc 0");
            counter = 0;
        }
    }
}

```

1. Tabs in the editor area indicate the names of resources that are currently open for editing. An asterisk (*) indicates that an editor has unsaved changes.
2. The Quick Diff feature displays color-coded indication for additions, deletions, or changes made to the contents of a file.
3. The marker bar displays:
 - Breakpoints (Auto, Hardware, Software, Disabled).
 - Markers (bookmarks, warnings, tasks, indexers, errors).
4. Icons flag error, warning, task and bookmark markers. You can view extra information by placing the mouse cursor over the marker.

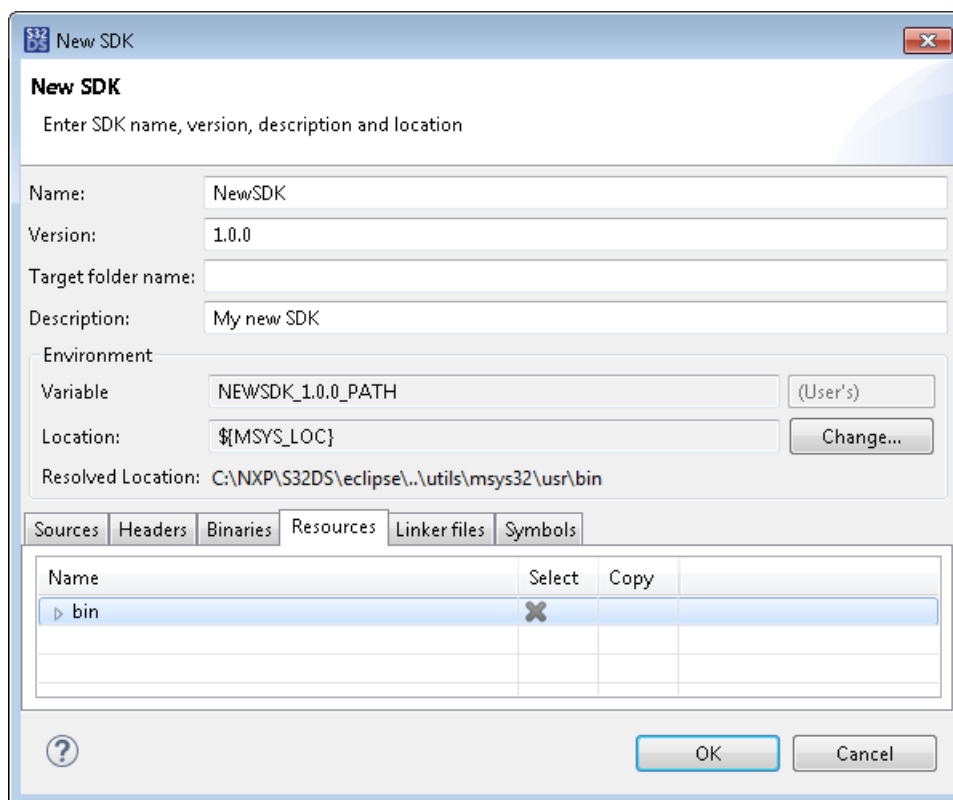
Tips and Tricks:

- To open the list of all open editors and quickly switch between them, press **Ctrl+E**.
- You can open a file in the editor area by dragging it from the **Project Explorer** view and dropping it over the editor area.
- To activate single-click opening for editors, use the **Open mode** options on the **Window > Preferences > General** page. In single-click mode, a single-click on a file selects and immediately opens the file.
- Double-clicking on the marker bar sets or removes breakpoints.
- To move lines up and down in the editor, press **Alt+Arrow Up** and **Alt+Arrow Down**.
- To activate code completion, press **Ctrl+Space**.
- To activate Quick Diff, right-click the marker bar and select **Show Quick Diff** from the context menu.
- To configure Quick Diff to use a different color code, click **Window > Preferences** on the menu and go to **General > Editors > Text Editors > Quick Diff** in the **Preferences** dialog box.
- When the mouse cursor is placed over a change in Quick Diff, a hover displays the original content, which can be restored using the marker bar context menu.

Wizards

New SDK wizard

The **New SDK** wizard serves for adding a third-party SDK to the workspace or to a project directly.



The following table describes the SDK properties:

Table 28: New SDK wizard properties

Property	Description
Name	The SDK name. Specify a valid name starting with a letter. Allowed characters: letters, numbers and underscores.
Version	The SDK version. Format: <major>.<minor>.<micro>.<qualifier>. “Major” is mandatory, other parts are optional. Allowed characters: digits (all parts), Latin letters and underscores (“major” only).
Target folder name	The SDK folder name in the project structure. After you attach the SDK to your project, the SDK files appear in the Project Explorer in the specified folder. Leave this field blank to use the SDK name for the project folder. Optional.
Description	A brief description of the SDK. Optional.
Variable	The environment variable that points the location of the SDK folder. By default, the variable is generated automatically from Name and Version .
Location	The path to the SDK folder stored in the variable. Click Change... to change the location of the SDK folder. Specify the path or point a different variable holding the path.
SDK files	<p>The SDK files available in the SDK folder. Categories: Sources, Headers, Binaries, Resources and Linker LD files.</p> <p>In each category, click SDK files to be used in destination projects. These files get a green “cross” mark.</p> <ul style="list-style-type: none"> Files marked in the Select column will be linked to the destination project. Files marked in the Copy column will be copied to the destination project. <p>The Symbols tab allows you to define or suppress the compiler and preprocessor symbols.</p>

Project creation wizards

Project creation wizards create a specific type of C or C++ project based on a project template for the chosen type. A project template contains factory settings and adds your project with default project files such as build scripts, launch configurations, and base implementations specific for the target hardware, source language, and type of target application (also known as build artifact).

A build artifact is the file produced as a result of a build. Each wizard sets the build artifact type based on the wizard type. For example, the application wizard configures the build artifact as ARM32 Executable. The library wizard that creates a project for a static library sets the build artifact as ARM32 Library.

Note: Changing the build artifact type after the project is created is not supported. Always use the proper project creation wizard to create the desired project.

Factory settings defined by project template configure toolchains for building and compiling code and define other metadata such as core preferences for the IDE. These settings include default configurations for the target processor core, source language, and debugger. For example, instead of configuring make files for the target C or C++ language, you can select the language in the project wizard, and the wizard will automatically fill MK files with necessary settings at build time. Settings available in project properties are defined by the project template and selected in the wizard.

The following project creation wizards are available:

- [S32DS Application Project wizard](#) - creates a project for a non-hosted C/C++ application.
- [S32DS Library Project wizard](#) - creates a project for a C/C++ non-hosted static library.
- [S32DS Project from Example wizard](#) - creates a project for based on a project example.

Wizards provide factory default settings and allow you to customize the settings by specifying project name, target processor family and making other changes specific to the target application or processor.

Default settings will be used in the created project if you leave their values intact and proceed with the wizard by clicking **Next** on its page. These settings can be further configured after creating the project by using project properties, see section *C/C++ Build Tool Settings*.

S32DS Application Project wizard

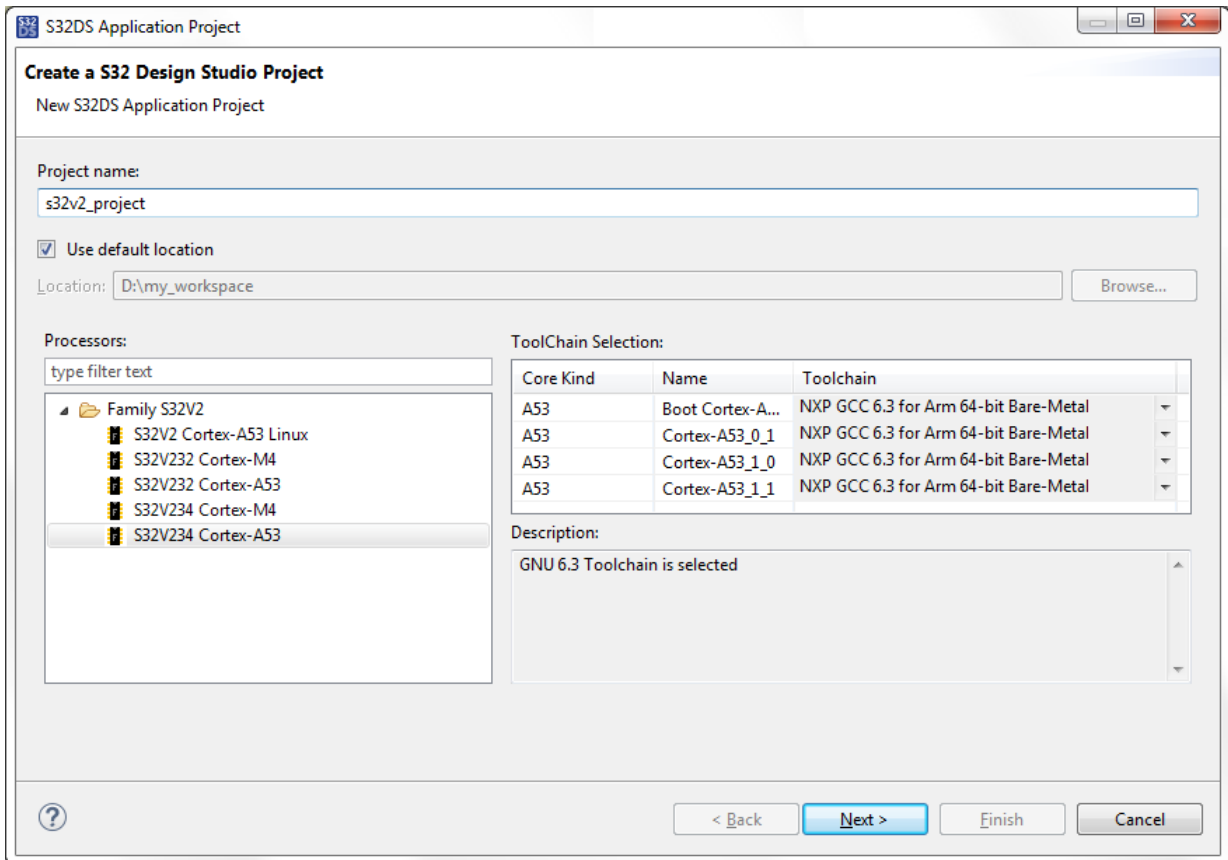
The **S32DS Application Project** wizard assists you in creating a new application project.

The availability of processors and cores depends on the installed packages.

Projects for non-hosted applications use Arm binary interfaces that allow your application to be compiled for SoC systems without an operating system on them. Such applications run standalone without using resources provided by the operating system layer. By default, the project wizard provides a minimum set of startup files, allowing the application to run on the target hardware without the OS layer.

General properties

The **Create a S32 Design Studio Project** page allows you to configure general properties of your project: the project name, the location where you want the project files to be stored, and the target processor and core on which you want the application to run.



The following table describes the settings that you can configure on this page.

Table 29: S32DS Application Project wizard: General properties

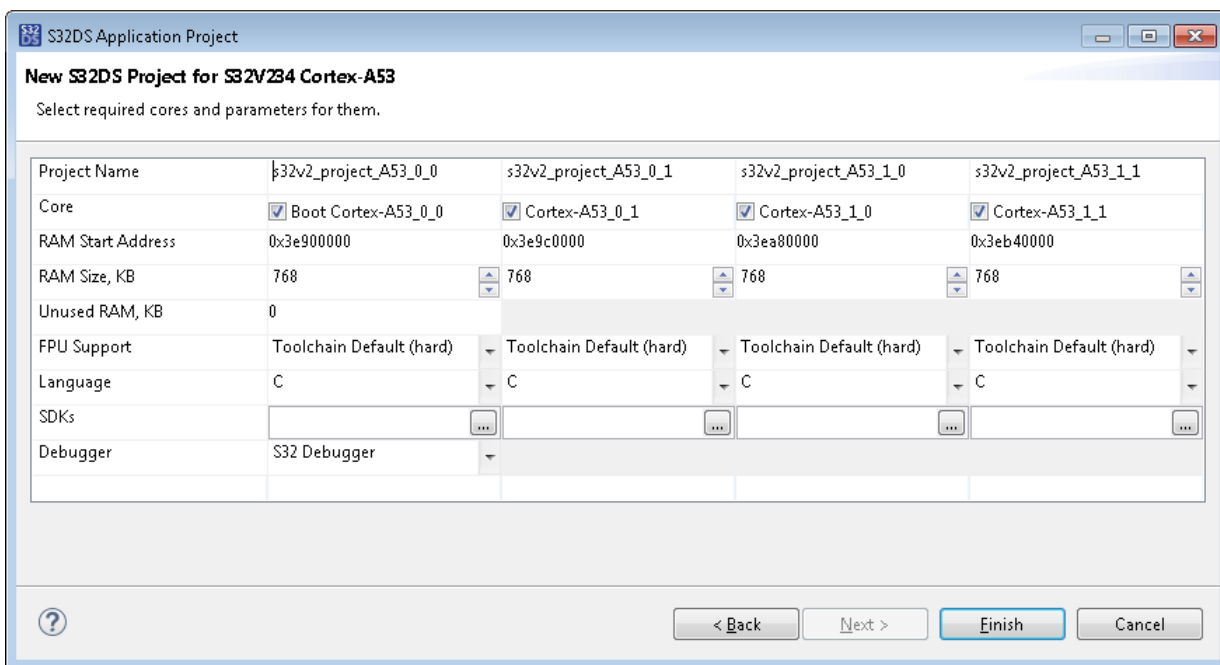
Option	Description
Project name	<p>The project name. Allowed characters: Latin letters (A-Z, a-z), digits (0-9), underscores. Do not start a project name with a digit.</p> <p>The project name is used at build time and must comply with standard C identifiers, symbols that you use in variables, function names, type definitions, and other namings in your application. If you create multiple projects within the same workspace, make sure to give unique names to projects that you include in this workspace.</p>
Use default location	<p>This option enables you to use the default workspace to store the project. By default, S32 Design Studio for S32 Platform stores project files in the current workspace. This location is displayed in the Location field.</p> <p>To specify a custom location, clear the check box and click Browse... to select a new location.</p>
Location	<p>The location where the project files will be stored.</p> <p>If Use default location is selected, this field is inactive and displays the default location. The Browse button is inactive.</p>
Processors	<p>The project type specific to target processor family and MCU where you want to use your application.</p>

Option	Description
ToolChain Selection	Details on the selected processor: core kind, core name, and GCC toolchain that will be used to build the project. Toolchain options can be further configured after creating the project, see the Build Tool Settings section.
Description	Brief information on the selected processor family or toolchain to be used.

Core customizations

The **New S32DS Project for <processor>** page allows you to customize the project properties so that the project could be built properly for the selected processor and core. You can specify the programming language, the I/O to be used, and the floating point support (hardware or software) to be used by the toolchain.

Note: The availability of properties depends on the selected processor. Some processors may not support certain properties.



The following table describes the settings that you can configure on this wizard page.

Table 30: S32DS Application Project wizard: Core customization settings

Option	Description
Project Name	The name assigned to the project on the General properties page of the wizard. Note: This name cannot be edited in-place. Click Back to specify a different name on the General properties page.
Core	The Arm® core used in the selected processor. Note: This check box cannot be cleared. Click Back to select a different processor on the General properties page.
RAM Start Address	The RAM Start Address values for each core. The values depend on the selected RAM Size.

Option	Description
RAM Size, KB	Specify the RAM Size value: from 0 to 1024 with step 32. The minimum size is 192 KB.
Unused RAM, KB	The Unused RAM value for the core. The value depends on the selected RAM Size.
Library	<p>The library to be linked to the application.</p> <p>Options:</p> <ul style="list-style-type: none"> • EWL - Embedded Warrior Library. • EWL Nano - a lightweight version of Embedded Warrior Library. • NewLib - standard C/C++ library. • NewLib Nano - a lightweight version of the NewLib library for minimalistic embedded applications that can dramatically reduce the size of your application.
I/O Support	<p>This setting enables the semihosting support and configures I/O to print information to the console. Options:</p> <ul style="list-style-type: none"> • No I/O - no printing will be done. • Debugger Console - the output will be printed out to the console provided by the debugger specified in the Debugger setting below.
FPU Support	<p>This setting enables GCC to build a project with the floating point support provided either by the processor or by a software library.</p> <p>The availability of options depends on the core used in the selected processor.</p> <p>Options:</p> <ul style="list-style-type: none"> • Toolchain Default - generation of floating-point instructions is defined by the FPU support in the selected processor. • Software: No FPU (-mfloat-abi=soft) - causes GCC to generate output containing library calls for floating-point operations. • Hardware: -mfloat-abi=hard - allows generation of floating-point instructions and uses FPU-specific calling conventions. • Hardware: -mfloat-abi=softfp - allows the generation of code using hardware floating-point instructions, but still uses the soft-float calling conventions. • Toolchain Default (hard) - generation of floating-point instructions is defined by the FPU support in the selected processor. If the FPU is enabled in the core, floating-point instructions are generated by the core and the hard-float calling conventions are used. • None - forces GCC to skip use of the FPU. • (not set) • No Floating-Point • Hardware Coprocessor • Hardware Single, Software Double • Software Emulation <p>Note: Find more information about Arm options in the GCC toolchain documentation on the web.</p>
Language	<p>This setting sets up the default compiler, linker, and preprocessor options for the toolchain, and configures other project files, such as main, for the target language.</p> <p>Note: The selected programming language defines the toolchain settings for the linker and compiler that will be available in the properties for the created project. Selecting C limits the toolchain options to this specific language. If you select C++, you will be able to configure settings for the C and C++ compiler, linker, and preprocessor. The toolchain</p>

Option	Description
	<p>settings can be further configured after creating the project. For details, see section "C/C++ + Build Tool Settings".</p> <p>Options:</p> <ul style="list-style-type: none"> • C - sets up your project for the ANSI C-compliant startup code and initializes global variables. • C++ - sets up your project for the ANSI C++ startup code and performs the global class object initialization.
SDKs	<p>This setting allows you to select an SDK to be added to the project. Click the search button (...) to select an SDK from the list.</p> <p>Note: The Select SDK window lists the SDKs available in S32 Design Studio for S32 Platform. If you do not see your SDK, add it on the SDK Management page. Find the details in Adding an SDK.</p> <p>Default: SDK is not selected.</p>
Debugger	<p>The debugger client to be used. Options:</p> <ul style="list-style-type: none"> • S32 Debugger • GDB PEMicro Debugging Interface • Lauterbach T32 Debugging Interface • GDB Remote C/C++ Application Debugger • VDK Debugging Interface • VLAB Simulator • Segger Debugging Interface • iSystem Debugging Interface • IAR Debugging Interface

S32DS Library Project wizard

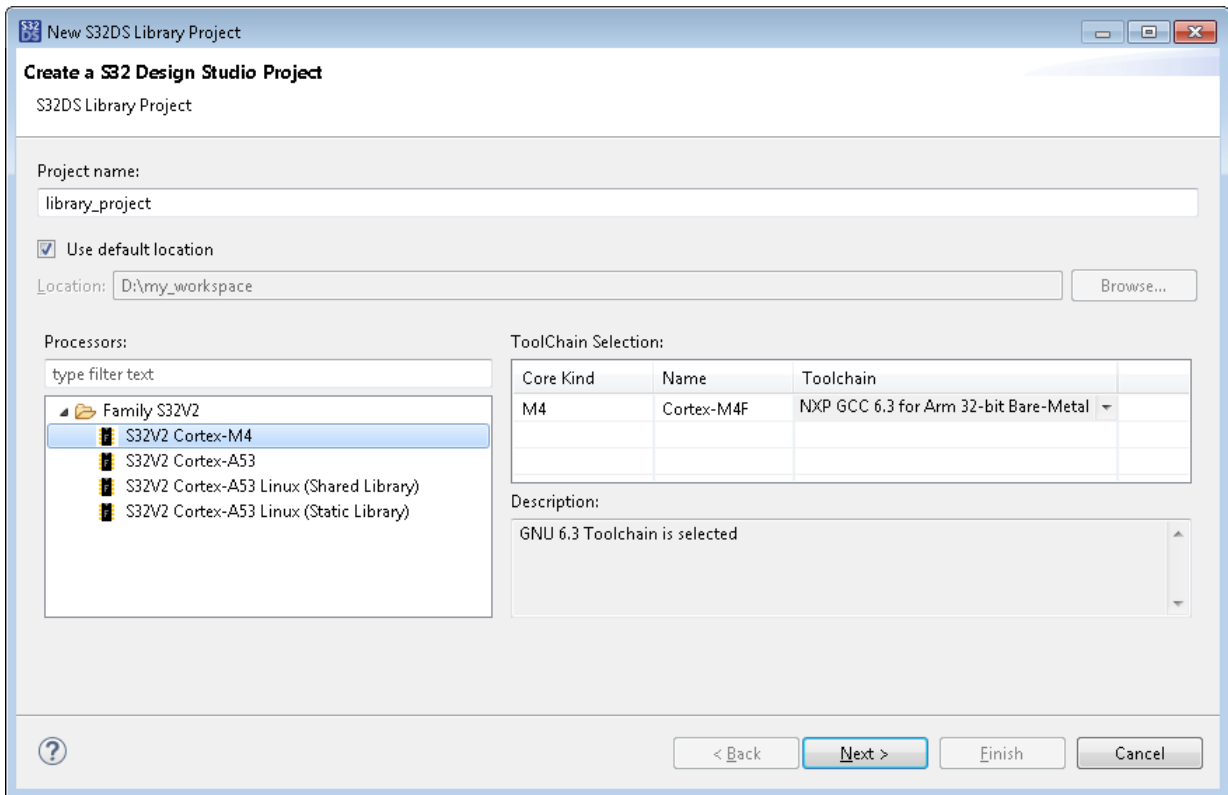
The **S32DS Library Project** wizard assists you in creating a new library project.

The availability of processors and cores depends on the installed packages.

Projects for libraries for non-hosted applications allow your library to be statically linked to application compiled for SoC systems that do not have any operating system on them. Such applications run standalone without using resources provided by operating system layer. Because there are no file system and operating system on the target SoC system, libraries for non-hosted applications are loaded into SoC memory and are statically linked to the standalone application.

General properties

The **Create a S32 Design Studio Project** page allows you to configure general properties of your project: the project name, the location where you want the project files to be stored, and the target processor and core on which the library will be linked by an embedded program.



The following table describes the settings that you can configure on this page.

Table 31: S32DS Library Project wizard: General properties

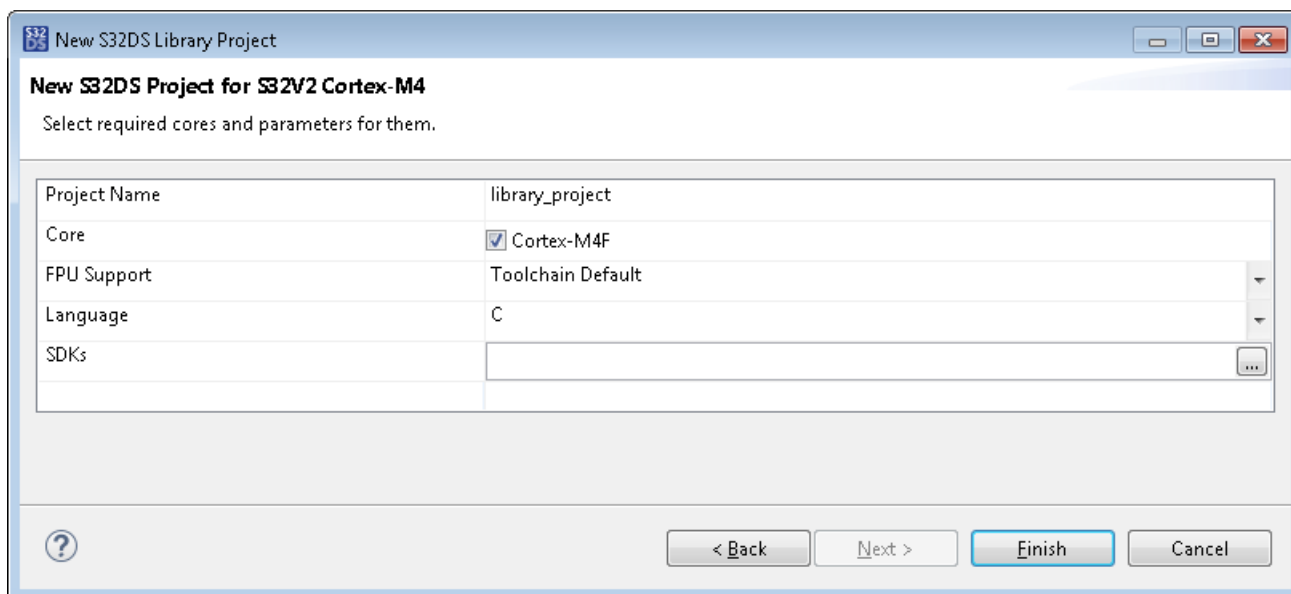
Option	Description
Project name	<p>The project name. Allowed characters: Latin letters (A-Z, a-z), digits (0-9), underscores. Do not start a project name with a digit.</p> <p>The project name is used at build time and must comply with standard C identifiers, symbols that you use in variables, function names, type definitions, and other namings in your library. If you create multiple projects within the same workspace, make sure to give unique names to projects that you include in this workspace.</p>
Use default location	<p>This option enables you to use the default workspace to store the project. By default, S32 Design Studio for S32 Platform stores project files in the current workspace. This location is displayed in the Location field.</p> <p>To specify a custom location, clear the check box and click Browse... to select a new location.</p>
Location	<p>The location where the project files will be stored.</p> <p>If Use default location is selected, this field is inactive and displays the default location. The Browse button is inactive.</p>
Processors	<p>The project type specific to target processor family and MCU where you want to use your library.</p>
ToolChain Selection	<p>Details on the selected processor: core kind, core name, and GCC toolchain that will be used to build the project.</p>

Option	Description
	Toolchain options can be further configured after creating the project, see the Build Tool Settings section.
Description	Brief information on the selected processor family or toolchain to be used.

Core customizations

The **New S32DS Project for <processor>** page allows you to customize the project properties so that the project could be built properly for the selected processor and core. You can specify the programming language, the I/O to be used, and the floating point support (hardware or software) to be used by the toolchain.

Note: The availability of properties depends on the selected processor. Some processors may not support certain properties.



The following table describes the settings that you can configure on this page.

Table 32: S32DS Library Project wizard: Core customization settings

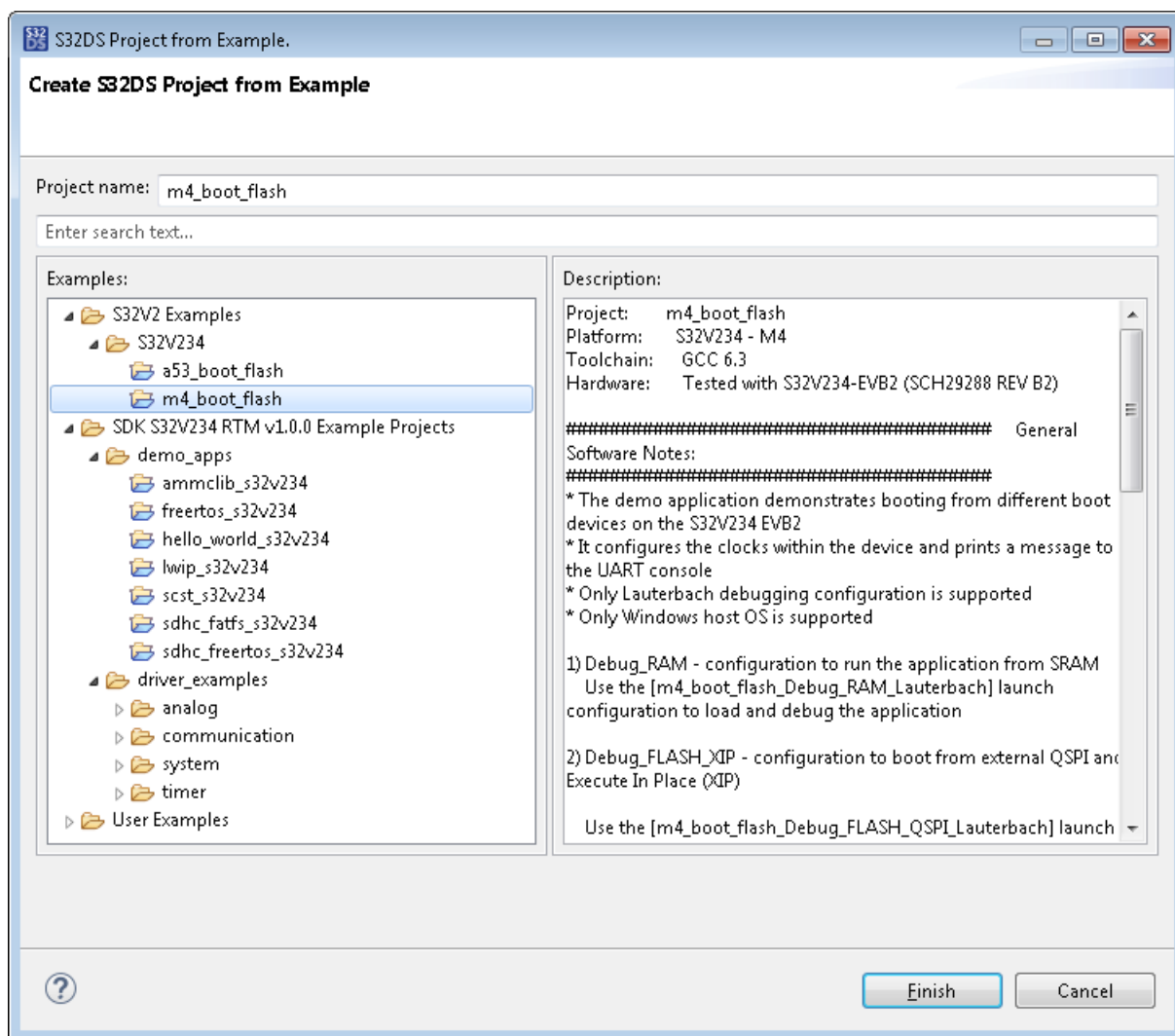
Option	Description
Project Name	The name assigned to the project on the General properties wizard page. Note: This name cannot be edited in-place. Click Back to specify a different name on the General properties page.
Core	The Arm® core used in the selected processor. Note: This check box cannot be cleared. Click Back to select a different processor on the General properties page.
FPU Support	This setting enables GCC to build a project with the floating point support provided either by the processor or by a software library. The availability of options depends on the core used in the selected processor. Options : <ul style="list-style-type: none"> • Toolchain Default - generation of floating-point instructions is defined by the FPU support in the selected processor.

Option	Description
	<ul style="list-style-type: none"> • Software: No FPU (-mfloat-abi=soft) - causes GCC to generate output containing library calls for floating-point operations. • Hardware: -mfloat-abi=hard - allows generation of floating-point instructions and uses FPU-specific calling conventions. • Hardware: -mfloat-abi=softfp - allows the generation of code using hardware floating-point instructions, but still uses the soft-float calling conventions. • Toolchain Default (hard) - generation of floating-point instructions is defined by the FPU support in the selected processor. If the FPU is enabled in the core, floating-point instructions are generated by the core and the hard-float calling conventions are used. • None - forces GCC to skip use of the FPU. <p>Note: Find more information about Arm options in the GCC toolchain documentation on the web.</p>
Language	<p>This setting sets up the default compiler, linker, and preprocessor options for the toolchain and configures other project files, such as main, for the target language.</p> <p>Note: The selected programming language defines the toolchain settings for the linker and compiler that will be available in the properties for the created project. Selecting C limits the toolchain options to this specific language. If you select C++, you will be able to configure settings for the C and C++ compiler, linker, and preprocessor. The toolchain settings can be further configured after creating the project. For details, see section "C/C++ + Build Tool Settings".</p> <p>Options:</p> <ul style="list-style-type: none"> • C - sets up your project for the ANSI C-compliant startup code and initializes global variables. • C++ - sets up your project for the ANSI C++ startup code and performs the global class object initialization.
SDKs	<p>This setting allows you to select an SDK to be added to the project. Click the search button (...) to select an SDK from the list.</p> <p>Note: The Select SDK window lists the SDKs available in S32 Design Studio for S32 Platform. If you do not see your SDK, add it on the SDK Management page. Find the details in Adding an SDK.</p> <p>Default: SDK is not selected.</p>

S32DS Project from Example wizard

The **S32DS Project from Example** wizard creates a new project on the basis of a project sample. The new project includes all files of the sample project.

To launch the **S32DS Project from Example** wizard, click **File > New > S32DS Project from Example** on the menu bar.

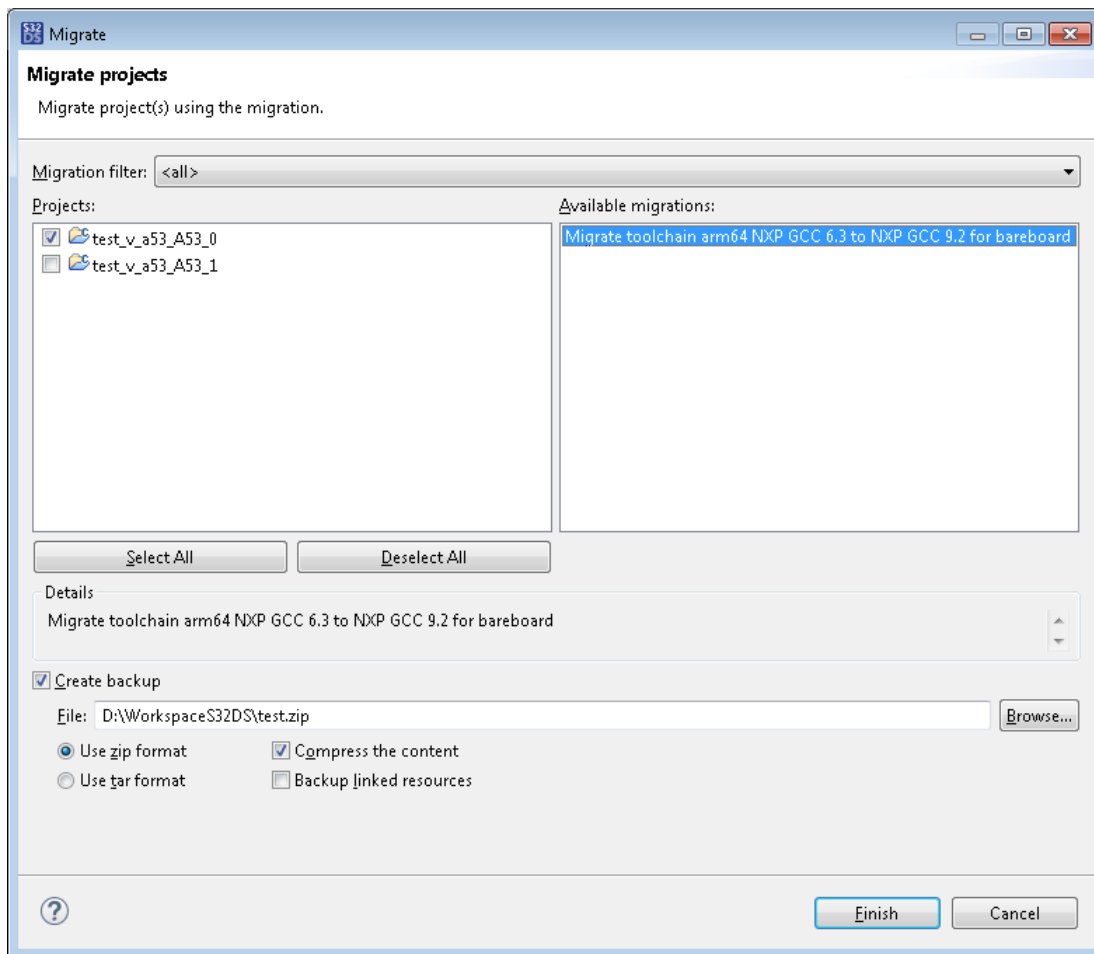


UI control	Description
Examples pane	Displays sample projects arranged in sections. The User Examples section is empty by default. This section is reserved for sample projects created by the user. Find the details in Saving a project to User Examples . Projects in the remaining sections are installed with S32 Design Studio for S32 Platform.
Description pane	Displays the information about the project currently selected in the Examples pane. This information may be missing.
Search box	Serves for quick search in the Examples pane.
Project name field	Automatically displays the name of the selected sample project. The displayed name can be edited. Note: The field is empty and cannot be edited until a sample project is selected in the Examples pane.
Finish button	Saves a copy of the selected project with the specified project name. The new project is displayed in the workspace for editing and debugging.

Migrate wizard

The **Migrate** wizard assists you in migrating projects to new SDK or toolchain available.

The availability of migrations depends on the installed packages.



The following table describes the settings that you can configure.

Table 33: Migrate wizard properties

Option	Description
Migration filter	Allows to sort out projects from the workspace with the selected migration available. Options: <ul style="list-style-type: none"> • <all> • Migrate SDK for <processor> from [X.X.X] to [Y.Y.Y] • Migrate toolchain armXX NXP GCC 6.3 to NXP GCC 9.2 for bareboard
Projects	Shows the available projects depending on migration type selected.
Available migrations	The availability of migrations depends on Projects selection (project type based) and Migration filter selection.
Details	Brief information on the selected migration to be performed.
Create backup	Enable this option to create a backup of your project.

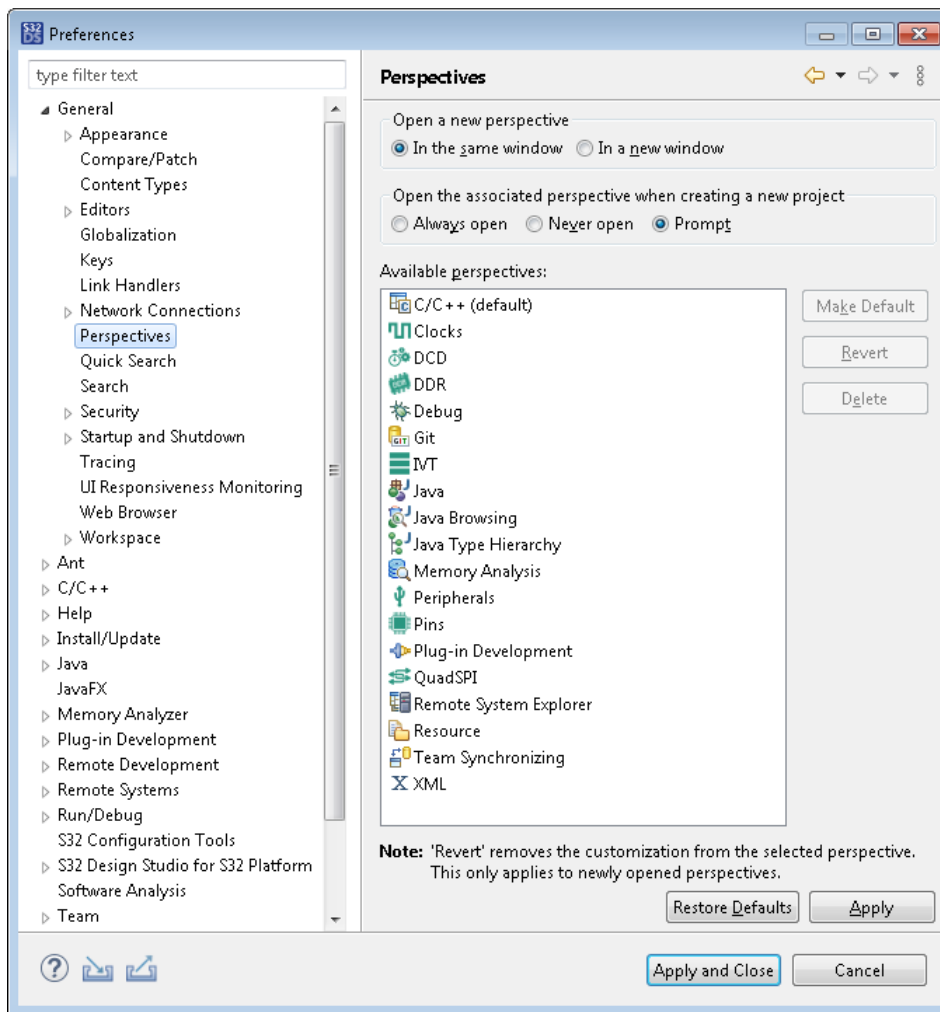
Option	Description
File	The location where the project backup will be stored. Click Browse button to select location and filename. If Create backup checkbox is clear, this field is inactive. The Browse button and backup properties selection are inactive.
Backup format	Backup archive type selection. Options: <ul style="list-style-type: none"> • Use zip format • Use tar format
Compress the content	Enable this option to compresses the contents in the archive that is created.
Backup linked resources	Enable this option to backup project with all linked resources.

Preferences

Perspectives

To configure the behavior of perspectives in the workbench, click **Window** > **Preferences** on the menu. In the **Preferences** dialog box, click **General** and **Perspectives**.

The **Perspectives** page allows you to define how perspectives will be opened in the workbench:



If the **In the same window** option is set, the selected perspective appears in the main window, hiding the previously displayed perspective. If the **In a new window** option is set, a new window is created for each opened perspective.

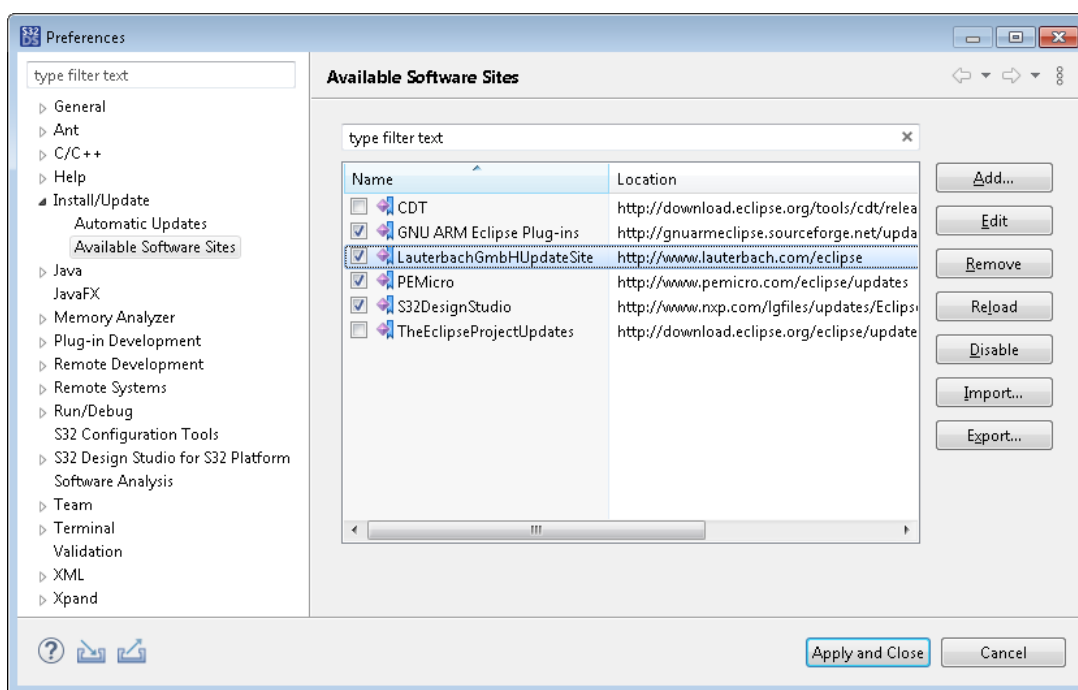
The **Available perspectives** list displays all perspectives, both predefined and created by the user, that can be opened in the workbench. Select a perspective from the list and manage it using the buttons located at the right border of the dialog box:

- **Make Default:** Click to use the selected perspective by default.
- **Revert:** Click to restore the initial layout and configuration settings for the selected perspective. This action is applicable to perspectives added by the user.
- **Delete:** Click to delete the selected perspective from the list. This action is applicable to perspectives added by the user.

Available software sites

S32 Design Studio for S32 Platform provides a tool to help you find and install the latest product updates and additional software packages. The lookup is performed across the sites that are specified in the user preferences.

To preview and edit the list of software sites, click **Window > Preferences** on the menu. In the **Preferences** dialog box, click **Install/Update** and **Available Software Sites**.



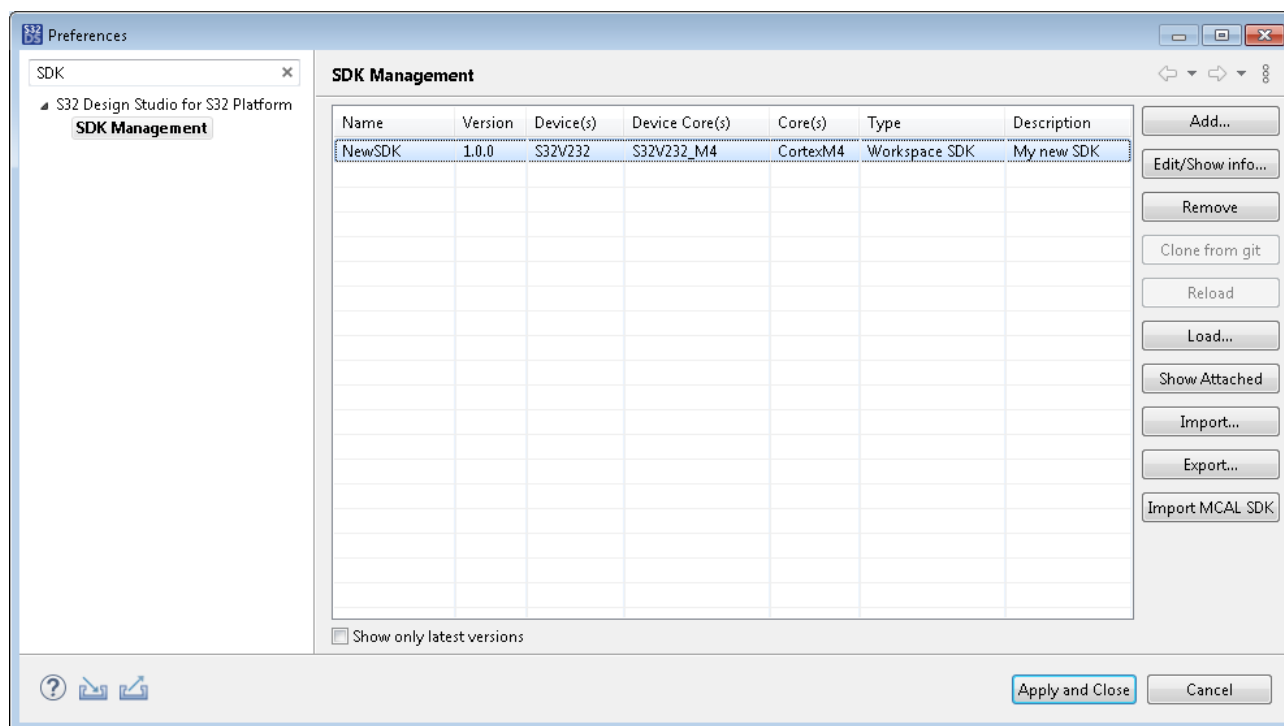
The right pane displays the list of software sites that are scanned for available updates automatically. You can manage the software sites using the buttons located at the right border of the dialog box:

- **Add:** Click to add a new site. Specify a custom name of the site and the location - a URL, a local directory, or a locally stored archive file.
- **Edit:** Click to modify the name and location of the selected site. To select a site, click it in the list. The selected site is highlighted.
- **Remove:** Click to remove the selected site from the list.
- **Reload:** Click to load the information about available updates from the selected site.
- **Enable/Disable:** Click to flag the selected site, or to remove the flag from the selected site.
- **Import:** Click to add sites from the specified XML file.
- **Export:** Click to export flagged sites to an XML file.

SDK Management

The **SDK Management** page in **Windows > Preferences > S32 Design Studio for S32 Platform** allows you to manage SDKs. **SDK Management** contains the list of all available SDKs except local ones, created using the [SDKs](#) page in the project properties.

Note: Some columns may appear or disappear depending on defined fields in all SDKs available (e.g. the **Core(s)** column appears only if some SDK has that field defined).



The following table lists options available on the **SDK Management** page. Some buttons can be disabled depending on the selected SDK type.

Table 34: Preferences: SDK Management

Option	Result
Add...	Creates a new SDK.
Edit/show info...	Modifies the properties of the existing SDK (only when the SDK is not attached to any project) or displays the properties if the selected SDK is read-only.
Remove	Removes the selected SDK.
Clone from git	Clones the SDK content from Git. The repository must have the SDK descriptor (the <code>sources.xml</code> file) in the root.
Reload	Updates the SDK content according to the latest changes in the SDK descriptor.
Load...	Adds the SDK using the XML descriptor.
Show Attached	Shows the list of projects to which the SDK is attached.
Import...	Imports the SDK from an archive file or from a directory.
Export...	Exports the selected SDK to an archive file.
Import MCAL SDK	Creates a new SDK on the basis of the MCAL SDK.
Show only latest versions	Select this check box to see only the latest versions.

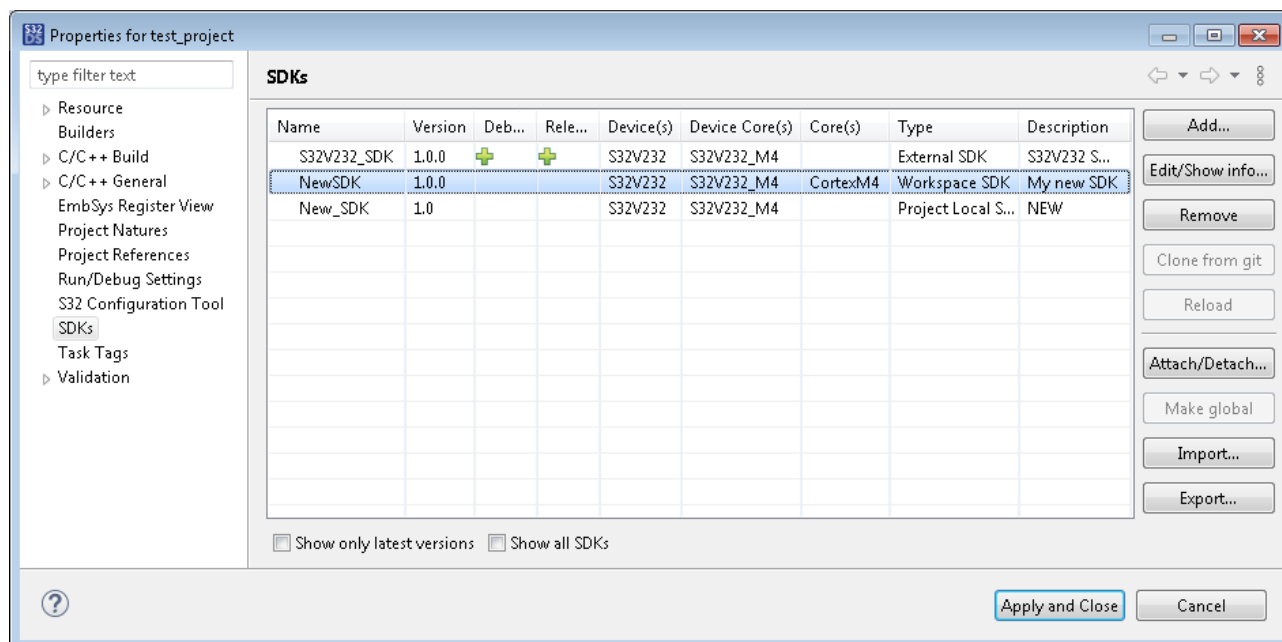
Project properties

SDKs

The **SDKs** page of the project properties displays information about available SDKs that are compatible with the project by the following criteria:






- supported compiler(s)
- supported language (C only or C/C++)
- supported architecture/core – if the software module is independent from peripherals and depends only on the core type
- supported operating system
- supported target device (core) – if the software module uses some hardware modules then it could be used only for a certain device (or core in case of a multi-core device).

Note: Some columns may appear or disappear depending on defined fields in all SDKs available (e.g. the **Core(s)** column appears only if some SDK has that field defined).



If a given SDK is attached to a project build configuration, the **+** mark is displayed in the respective column.

By default, the page displays SDKs that are attached or can be attached to the project. If you select the **Show all SDKs** option, the incompatible SDKs may be displayed as well (if such exist in the workspace). The following marks in the build configuration columns indicate the problem:

-  The toolchains of the project and SDK are incompatible
-  The processor or core of the project and SDK are incompatible
-  The languages of the project and SDK are incompatible
-  The project root folder contains the file with the name matching the SDK name
-  The SDK folder cannot be found (renamed or deleted)



The host operating system and SDK are incompatible



The project root folder contains the folder with the name matching the SDK name. The comparison is case-insensitive.

The following buttons and options are available for managing SDKs in a project:

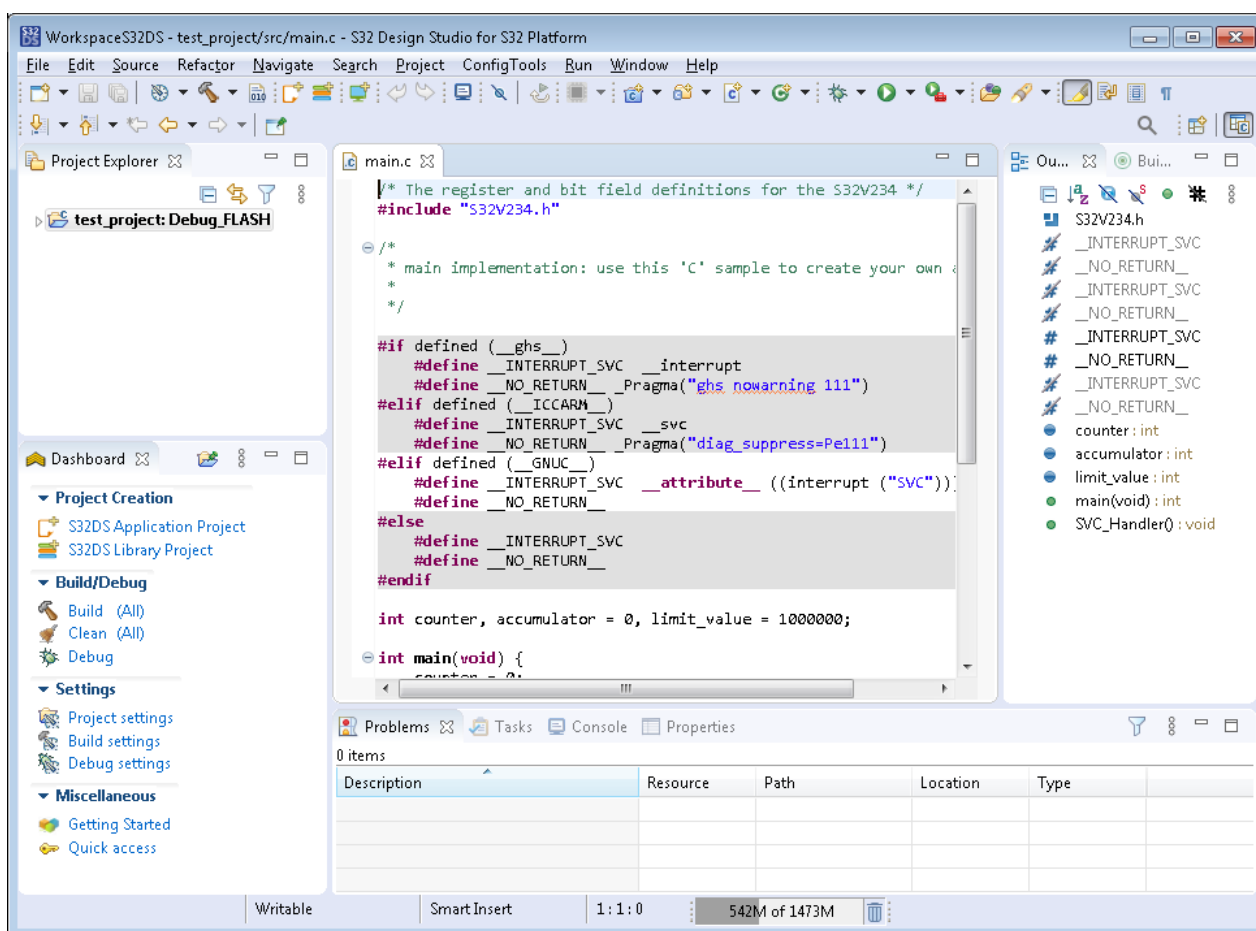
Table 35: Project properties: SDKs

Button/Option	Result
Add...	Creates a new local SDK. The SDK will be available only for the current project.
Edit/Show info...	Modifies properties of the existing SDK (when the SDK is not attached to any project) or displays the properties if the selected SDK is read-only.
Remove	Removes the selected SDK.
Clone from git	Clones the SDK content from Git. The repository must have the SDK descriptor (the sources.xml file) in the root.
Reload	Updates the SDK content according to the latest changes in the SDK descriptor.
Attach/Detach	Attaches/detaches the SDK to/from the project and uses it in the specified build configuration.
Make global	Makes the selected local SDK global. The SDK will be available in the SDK Management list.
Import...	Imports the SDK from an archive file or from a directory.
Export...	Exports the selected SDK to an archive file.
Show only latest versions	Select this option to see only the latest version of each SDK.
Show all SDKs	Select this option to see all global SDKs that cannot be attached to the project.

Perspectives

C/C++ perspective

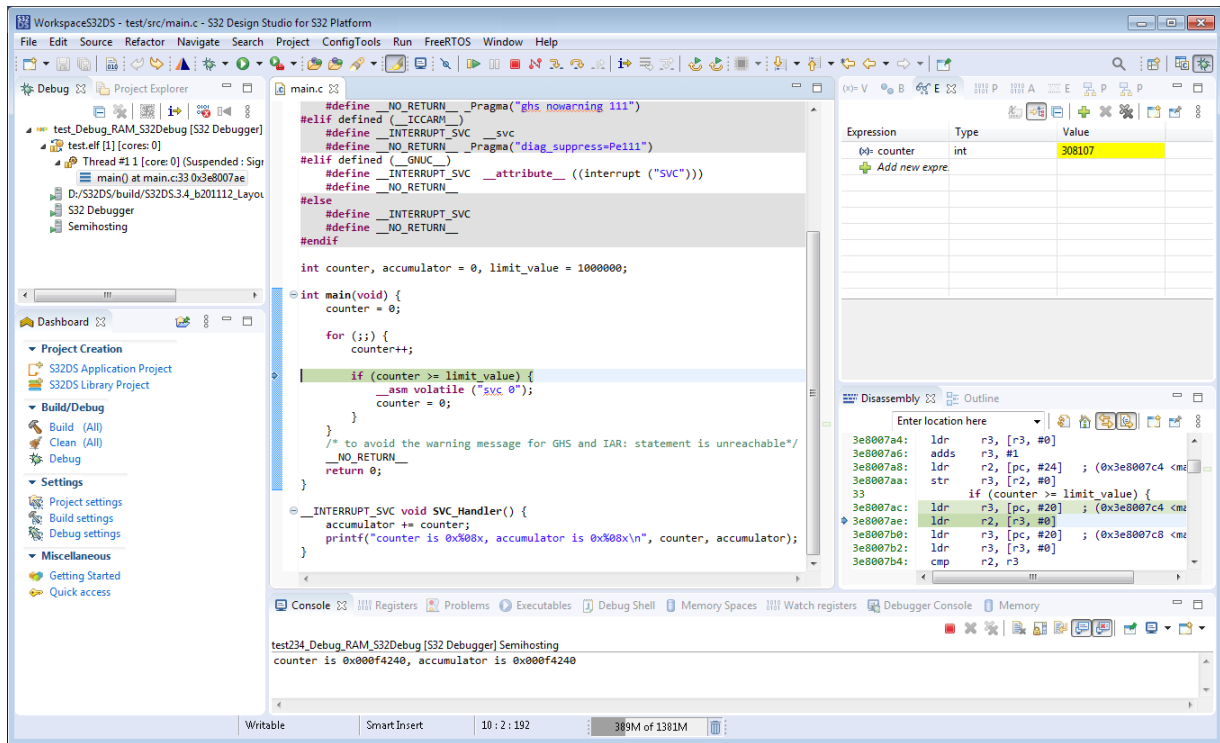
The C/C++ perspective is used for designing C and C++ projects.



- The **Project Explorer** view provides the hierarchical view of project resources.
 - To open a source file for editing, double-click it in the **Project Explorer** view.
 - Right-click any resource in the **Project Explorer** view to open the context menu that allows you to perform operations such as copying, moving, creating new resources, comparing resources with each other, or performing team operations.
 - To quickly import files and folders to your project, drag them from the system folder to the **Project Explorer** view.
 - Similarly, to export files and folders, drag them from the **Project Explorer** view to the system folder.
- The **Dashboard** view provides quick access to some basic features and frequently used functions.
- The *editor area* enables you to open and modify project files. For details, see [Editor area](#).
- The **Outline** view displays the outline of the file opened in the editor area.
- The **Problems** view lists the compilation errors and the files where these errors have occurred. Click an error in the **Problems** view to open the associated file in the editor. The cursor and the highlighted text indicate the line of code where the error has been encountered.

Debug perspective

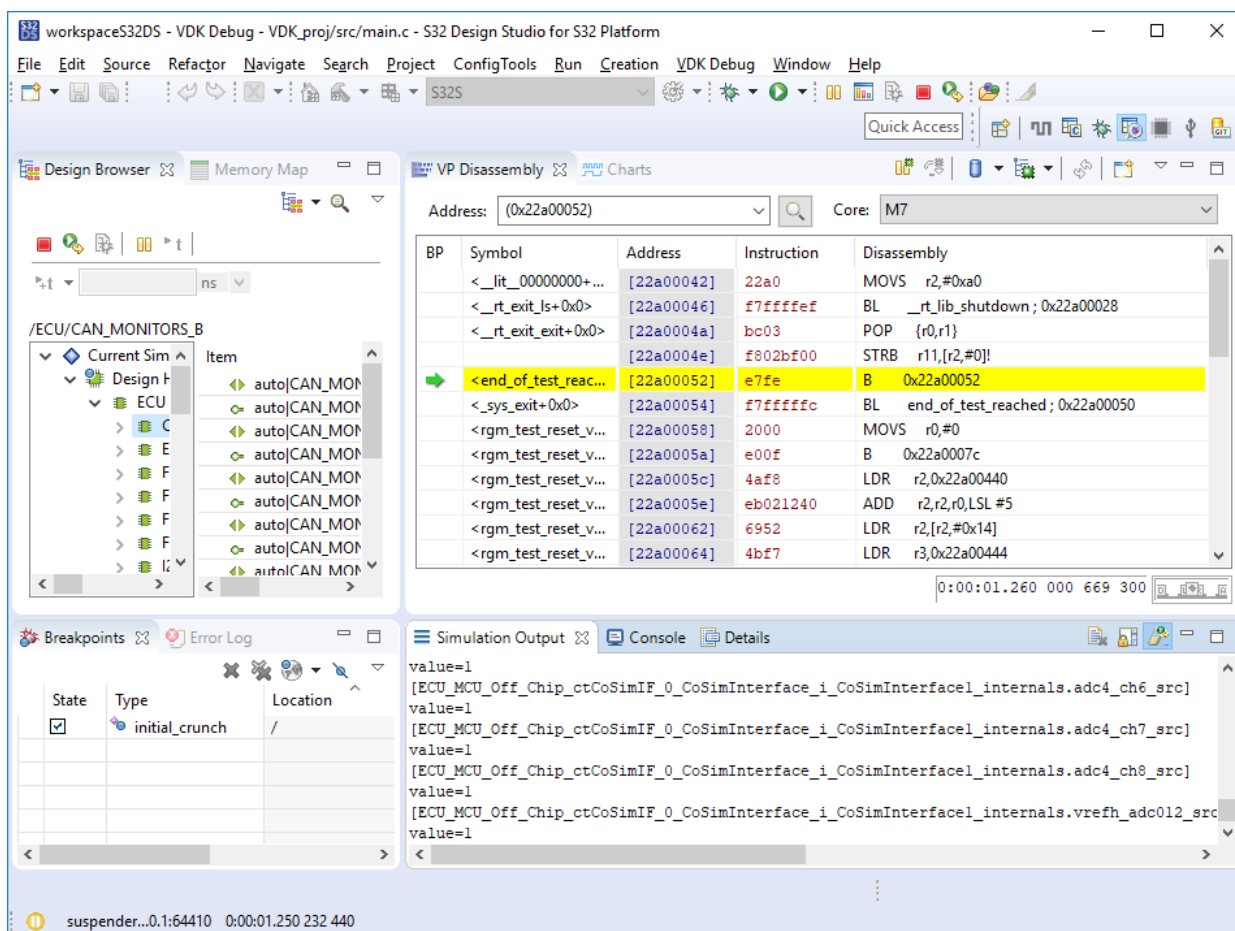
The Debug perspective enables you to manage the debugging or running of a program. You can control the execution of your program by setting breakpoints, suspending threads, stepping through the code, and examining the values of variables.



- The **Debug** view displays the hierarchy of debug instances and allows you to manage the debugging or running of a program.
 - The root node of the hierarchy is the process on the target on which you are debugging.
 - The nested nodes below the root represent the threads in the program.
 - If a thread is suspended, the stack frame appears in the nested nodes below the thread node.
- The **editor area** allows you to modify the contents of files. For more details, see [Editor area](#).
- The **Dashboard** view provides quick access to some basic features and frequently used functions.
- The **Variables** view shows all static variables for each process that you debug. Use the view to observe changes in the variables values as the program executes in the selected stack frame.
- The **Disassembly** view the loaded program as assembly language instructions mixed with source code for comparison.
- The **Console** view displays the output of the process and allows you to provide keyboard input to the process. There are numerous consoles that can be opened in the **Console** view. On the toolbar view, click the **Display Selected Console** and **Open Console** buttons to see all consoles available to you.

VDK Debug perspective

The VDK Debug perspective becomes available after the Synopsys simulation tools are installed. This perspective presents the user interface of the Synopsys VP Explorer tool.



The **Design Browser** view displays the hierarchy of module instances and tracks the state of a running simulation. The view communicates directly with the simulator when updating the states of modules, ports, and signals. Whenever the simulation is suspended, the view displays the actual values of ports, signals, and variables in a tree structure.

- The left pane of the view displays all module instances in the current simulation.
- The right pane of the view displays all members of the module instance selected in the left pane:
 - Ports and exports
 - Memories and registers
 - Signals and other primitive channels
 - Processes (SystemC methods and threads)
 - Member variables
 - Other SystemC objects
 - Monitors that have been attached to the design

The **Breakpoints** view tracks the list of breakpoints set by the user:

- The *State* column allows you to enable or disable the selected breakpoint. The context menu opened on the selected row offers options to manage the state of the selected breakpoint or all breakpoints.
- The *Type* column describes the type of the selected breakpoint. This type is also shown in the simulation location of the status bar when the breakpoint is hit. The tooltip on the column header shows the summary of all supported types of breakpoints.
- The *Location* column displays the full path of the object for which the breakpoint is defined. Global breakpoints show "/" (slash) as the path. The location is also shown in the simulation location of the status bar when the breakpoint is hit.

- The *Hit Count* column displays the number of breakpoint hits since the creation of the breakpoint.
- The *Ignore count* column displays the number of remaining hits after which execution will be stopped. The value can be adjusted. The value of zero is used by default and means that execution will stop next time the breakpoint is hit.
- The *TCL callback* column displays the Tcl command that will be executed when the simulation is stopped because of this breakpoint. The value can be entered and should be a valid Tcl expression. The default is no Tcl command executed for the breakpoint.

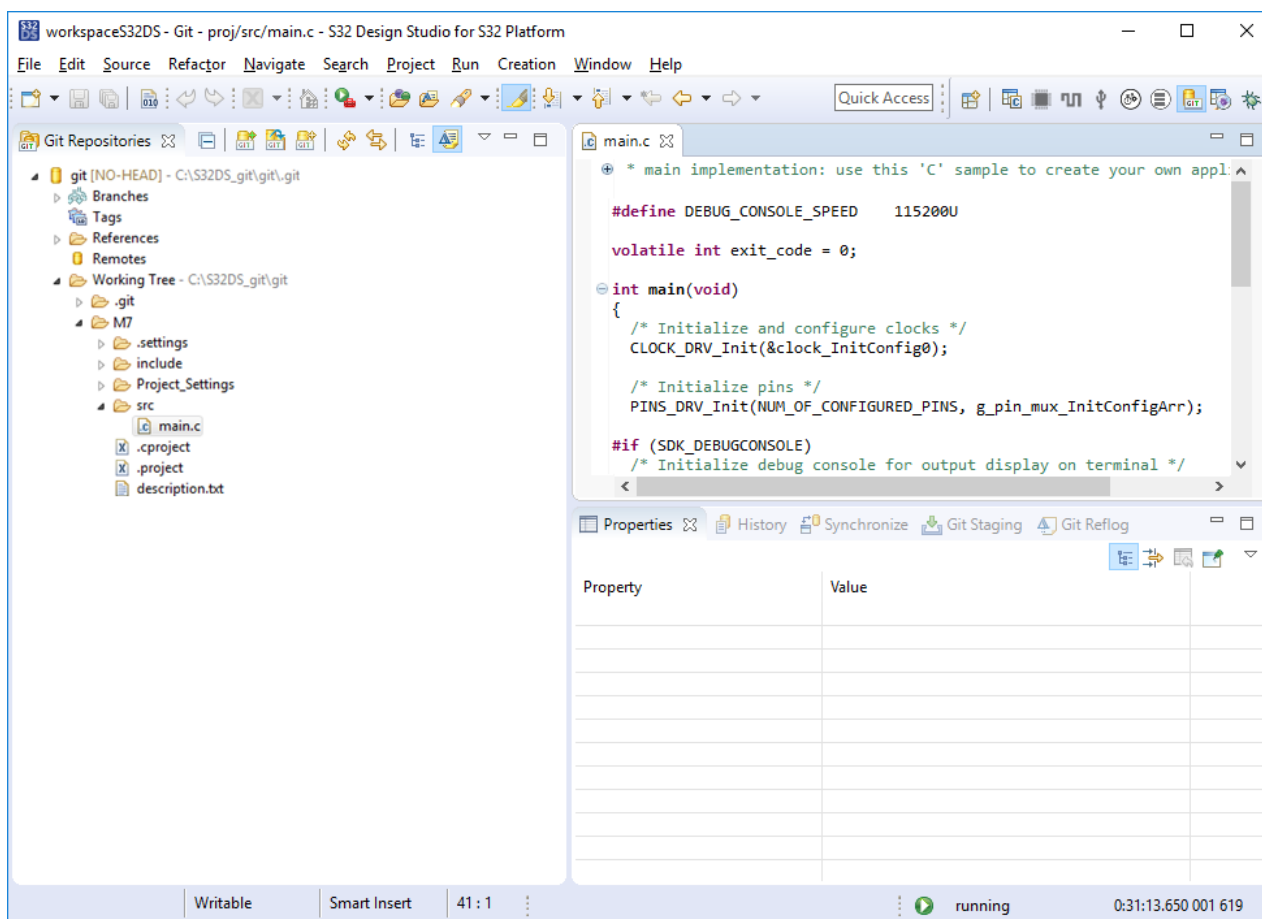
The **VP Disassembly** view displays the program currently executed on a virtual prototype representing the target processor. If the virtual prototype contains the model of the processor that has been instrumented, the **VP Disassembly** view shows the instructions being executed, with the associated disassembled code and symbols, and enables you to set breakpoints on the running program.

- The *Address* field indicates the value of the program counter in the selected core. The value is put in braces to indicate that it depends on the register.
- The *Core* field indicates the currently selected core.
- In the *BP* (“breakpoint”) column, the green arrow indicates the program counter.
- The *Symbol* column displays software labels (if any).
- The *Address* column displays addresses of the instructions.
- The *Instruction* column displays the hexadecimal version of the instructions.
- The *Disassembly* column displays the disassembled instructions.
- The status bar at the bottom of the view displays the simulation time for the core and the current context (process or thread) that runs on the core. The status of the core is shown by the icon at the right of the status bar.

The **Simulation Output** view displays the output of the running simulation. You can copy, paste and find strings in the output. By default, this view is cleared each time a new simulation is started. This behavior can be toggled off using the **Clear Simulation Output on simulation start** option.

Git perspective

The Git perspective provides the interface to Git operations.



The **Git Repositories** view displays the connected Git repository as a tree structure:

- The root node represents the repository itself. The node text indicates the name of the repository and its location in the local file system.
- The *Branches* node serves for browsing and manipulating tags.
- The *Tags* node serves for browsing and manipulating tags and branches.
- The *References* node lists references other than branches and tags, most notably the "HEAD" and "FETCH_HEAD" symbolic references.
- The *Remotes* node serves for browsing and manipulating remote configurations used for Fetch and Push.
- The *Working Tree* node displays the location and structure of the working directory in the local file system (only in case of a development, or non-bare repository; this node is always a leaf for bare repositories).

The *editor area* allows you to modify the contents of files. For details, see [Editor area](#).

The **History** view displays commits to the repository in the following panes:

- *Commit Graph* (upper pane): Displays the commit history in the reverse chronological order, with the newest commit displayed on top.
- *Revision Comment* area (left pane): Displays the commit message and a textual Diff of the file or files in the commit.
- *Revision Detail* area (right pane): Displays the table of files that were changed by the commit.

The **Git Staging** view displays the interface for staging and committing changes to the repository:

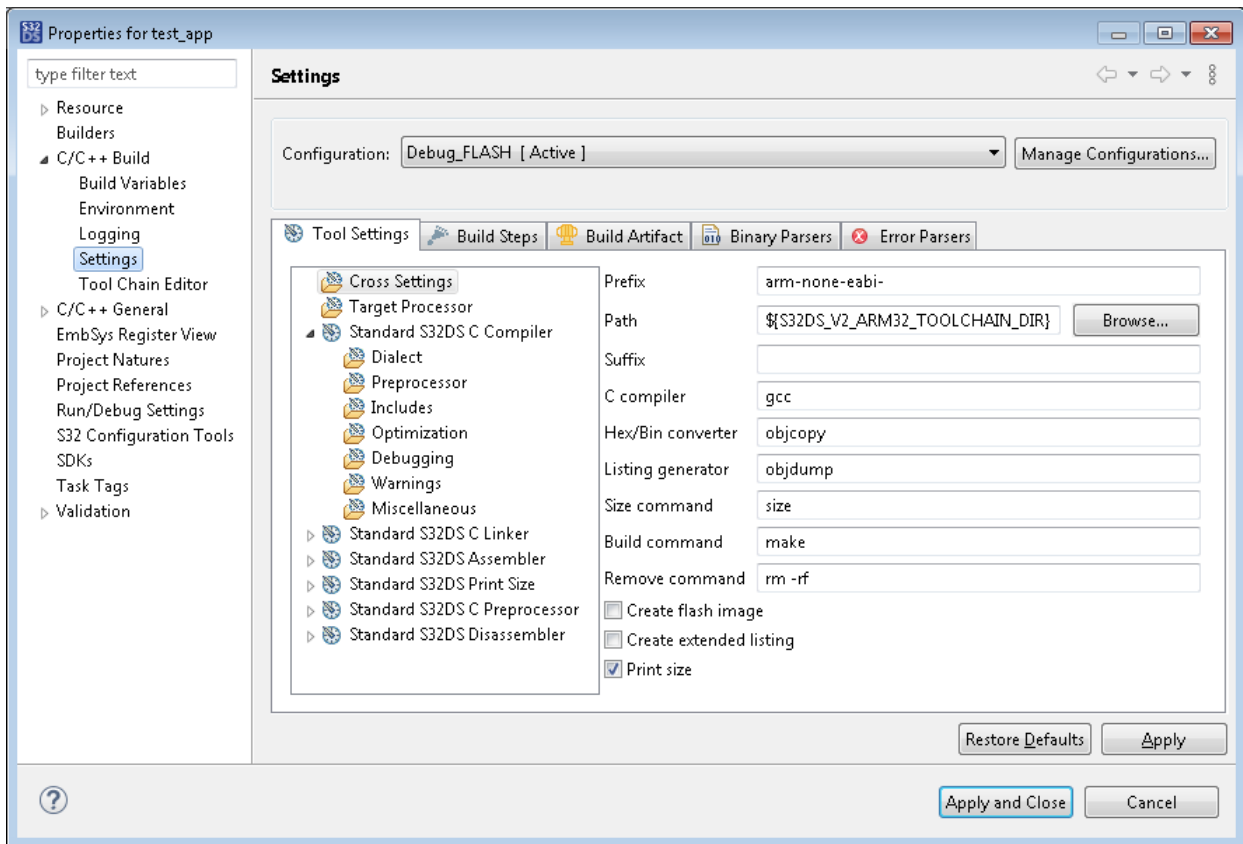
- *Unstaged Changes* pane: Displays the unstaged changes.
- *Staged Changes* pane: Displays the changes that have already been added ("staged") to the Git index.
- *Commit message* editor: Allows you to edit the commit message.
- *Commit* and *Commit and Push* buttons: Commit the staged changes.

Build configuration

Build Tool Settings

This section describes a collection of tool settings included in the *build configuration*. These tool settings are used by S32 Design Studio for S32 Platform when building projects. The tools are supplied in the GCC toolchain that you select when creating a new project.

To edit the tool settings, select a project in the **Project Explorer** and click **Project > Properties** on the main menu. In the **Properties** dialog box, click **C/C++ Build > Settings** on the left pane.



The **Tool Settings** tab on the right pane displays the tools that are described in this documentation section. To open the settings of a certain tool, click that tool in the tree view.

Cross Settings

The **Cross Settings** page displays the build configuration settings that apply to multiple or all tools in the toolchain.

The following table describes the cross settings available for application projects.

Table 36: Application Project Properties: Cross Settings

Setting	Description
Prefix	The toolchain prefix used to resolve names of the called tool.

Setting	Description
	Default: arm-none-eabi-, aarch64-none-elf- or aarch64-linux-gnu-.
Path	<p>The location of the toolchain. Options:</p> <ul style="list-style-type: none"> • $\\${S32DS_<device>_ARM32_TOOLCHAIN_DIR}$ • $\\${S32DS_<device>_ARM64_TOOLCHAIN_DIR}$ • $\\${S32DS_<device>_ARM64_LINUX_TOOLCHAIN_DIR}$ • $\\${S32DS_<device>_ARM32_GNU_9_2_TOOLCHAIN_DIR}$ • $\\${S32DS_<device>_ARM64_GNU_9_2_TOOLCHAIN_DIR}$ • $\\${S32DS_<device>_ARM64_GNU_9_2_LINUX_TOOLCHAIN_DIR}$ <p>Note: The build variables used in paths are available in the Preferences dialog box (section C/C++ > Build > Build Variables, the Show system variables option enabled).</p>
Suffix	The toolchain suffix used to resolve names of the called tool. Blank by default.
C compiler	<p>The C compiler executable.</p> <p>Default: gcc.</p> <p>The full name is resolved by adding the toolchain prefix.</p>
C++ compiler	<p>The C++ compiler executable.</p> <p>Default: g++.</p> <p>The full name is resolved by adding the toolchain prefix.</p>
Hex/Bin converter	<p>The HEX/BIN converter tool that copies and translates object files.</p> <p>Default: objcopy.</p> <p>This tool is used by the Standard S32DS Create Flash Image virtual tool.</p>
Listing generator	<p>The tool that displays information about object files.</p> <p>Default: objdump.</p> <p>This tool is used by the Standard S32DS Disassembler and Standard S32DS Create Listing virtual tools.</p>
Size command	<p>The tool that calculates (in bytes) the size of text, data and uninitialized sections in the ELF file, and their total.</p> <p>Default: size.</p>
Build command	<p>The tool that automatically builds executable programs and libraries from the project source code.</p> <p>Default: make.</p>
Remove command	<p>The tool and command parameters to remove the built executable programs, libraries, and object files.</p> <p>Default: rm -rf.</p>
Create flash image	<p>Select to enable the creation of a flash image at build time.</p> <p>The flash image is created from the built ELF file by the tool specified in the Hex/Bin converter setting. The <code><project_name>.srec</code> file appears in the Project Explorer in the project's Debug folder. The SREC file includes binary</p>

Setting	Description
	code in Motorola SREC text format that represents binary data as a hexadecimal text in ASCII format. You can flash this file to the target MCU. Enabling this option shows the Standard S32DS Create Flash Image tool in the list of tools.
Create extended listing	Select to enable the creation of extended listing at build time. The file is generated at build time by the tool specified in the Listing generator setting. The <project_name>.lst file appears in the Project Explorer in the project's Debug folder. The LST file contains the disassembly of the built ELF file and gives a bit more details on functions than the Standard S32DS Disassembler. Enabling this option shows the Standard S32DS Create Listing tool in the list of tools.
Print size	Select to call the size tool after the project is built. The tool outputs details to the Console view.

The following table describes the cross settings available for library projects.

Table 37: Library Project Properties: Cross Settings

Setting	Description
Prefix	The prefix used to call the tools. Default: arm-none-eabi-, aarch64-none-elf- or aarch64-linux-gnu-.
Path	The location of the toolchain. Options: <ul style="list-style-type: none"> • \${S32DS_<device>_ARM32_TOOLCHAIN_DIR} • \${S32DS_<device>_ARM64_TOOLCHAIN_DIR} • \${S32DS_<device>_ARM64_LINUX_TOOLCHAIN_DIR} • \${S32DS_<device>_ARM32_GNU_9_2_TOOLCHAIN_DIR} • \${S32DS_<device>_ARM64_GNU_9_2_TOOLCHAIN_DIR} • \${S32DS_<device>_ARM64_GNU_9_2_LINUX_TOOLCHAIN_DIR}
Suffix	The suffix used to call the tools. Blank by default.
C compiler	The C compiler executable. Default: gcc. The full name is resolved by adding the prefix.
C++ compiler	The C++ compiler executable. Default: g++. The full name is resolved by adding the prefix.
Archiver	The archiver tool. Default: ar. After you build a static library, the tool automatically archives the object file. The archive file appears in the Project Explorer in the project's Archives folder.
Build command	The tool that automatically builds executable programs and libraries from the project source code.

Setting	Description
	Default: <code>make</code> .
Remove command	The tool and command parameters to remove built executable programs, libraries, and object files. Default: <code>rm -rf</code> .

Target Processor

The **Target Processor** page displays the build configuration settings that apply to the target processor specified in an application project or in a library project.

In the following table, the Setting column lists the settings and the related GCC compiler options. If a setting has the “Toolchain default” option, this stands for the default (“factory”) GCC configuration setting. The availability of properties depends on the selected processor.

Table 38: Application and Library Project Properties: Target Processor

Setting	Description
Other target flags	Additional compiler options not included in the project properties. You can specify any required options supported by the compiler. Consult the compiler documentation. Important: For this setting to take effect, specify “Toolchain default” in the Target processor field.
Arm family	The target Arm [®] processor. GCC uses this name to derive the target Arm [®] architecture and the Arm [®] processor type to tune it for performance. Where this option is used in conjunction with <code>-march</code> or <code>-mtune</code> , those options take precedence over the appropriate part of this option. By default, the core specified in the project creation wizard is selected. Recommendations: <ul style="list-style-type: none"> Do not use options other than “cortex-m7”, “cortex-r52”, “cortex-m33” and “cortex-m4”. Other cores, though displayed, are not supported by the project creation wizard. If you select an unsupported core, you will have to manually recreate the project structure, the startup code and other metadata in the project. When you use the “Toolchain default” option, no core-specific options are passed to the compiler. Using this option is not recommended as the default option set may change without further notice.
Architecture <code>-march=<i>name</i></code>	The target Arm [®] architecture. GCC uses this name to determine what kind of instructions it can emit when generating assembly code.
Target processor <code>-mcpu={cortex-a53, cortex-a53+nofp}</code>	The target processor that will execute the code. Options: <ul style="list-style-type: none"> Toolchain default: Compiles the code by using the default core that was used to build GCC. cortex-a53: Compiles the code for Cortex-A53 with hardware support for floating-point instructions that will be performed by the integrated FPU. cortex-a53+nofp: Compiles the code for Cortex-A53 with no support for floating-point instructions.
Optimize	The target host processor for which you want to run code optimizations.

Setting	Description
<code>-mtune=name</code>	
Instruction set <code>-marm, -mthumb</code>	<p>The assembler instruction set for generating code that executes in the Arm or Thumb state.</p> <p>Options: Toolchain default, Thumb (<code>-mthumb</code>), Arm (<code>-marm</code>).</p>
Thumb interwork <code>-mthumb-interwork</code>	<p>This option enables GCC to generate code that supports calls between the Arm and Thumb instruction sets.</p>
Endianness <code>-mlittle-endian</code>	<p>This setting enables you to generate code for a processor running in the little-endian mode.</p> <p>Options: Toolchain default, Little endian.</p> <p>Default: Toolchain default.</p> <p>Note: By default, GCC is configured to generate code for a processor running in the little-endian mode.</p>
Float ABI <code>-mfloat-abi={hard, soft, softfp}</code>	<p>The floating-point application binary interface (ABI) that the GCC will use when compiling the code. Notice that hard-float and soft-float ABIs are not link-compatible; you must compile your entire program with the same ABI, and link with a compatible set of libraries.</p> <p>Options:</p> <ul style="list-style-type: none"> • Toolchain default: Compiles your code by using the default (specified in the GCC) ABI for the target processor. The GCC will detect the floating-point operations support based on the support for FPU in the selected processor. On Cortex-A53 based processors, this defaults to <code>-mfloat-abi=hard</code> so that the core is responsible for floating-point operations and the FPU-specific calling conventions are used. • Library (soft): Enables GCC to generate code with library calls so that floating-point operations are emulated by the compiler and not the FPU on the processor. • Library with FP (softfp): Enables GCC to generate code with support for hardware floating-point instructions provided by the processor while using soft floating-point ABI calling conventions. • FP instructions (hard): Enables GCC to generate code with support for hardware floating-point instructions provided by the processor, and uses the ABI calling convention specific to the FPU on the processor. <p>Default: Toolchain default.</p>
FPU Type <code>-mfpu=name</code>	<p>The floating-point unit (FPU) or hardware emulation available on the target processor.</p> <p>This setting is only available if hardware or hardware emulated ABI option (FP instructions (hard) or Library with FP (softfp) respectively) is selected in Float ABI and is currently locked to the fpv5-sp-d16 architecture for handling floating-point operations. This architecture includes support for FP registers that can be used by your application as 32 single-precision floating point registers or as 16 double-precision floating point registers.</p> <p>Options: Toolchain default, fpv5-sp-d16.</p> <p>Default: Toolchain default. The GCC will select the floating-point instructions based on the settings specified in the Architecture and Target processor settings.</p>

Setting	Description
Unaligned access -munaligned-access, -mno-unaligned-access	<p>This option enables or disables access to addresses not aligned to 16 or 32 bits. If unaligned access is disabled, words in packed data structures are accessed a byte at a time.</p> <p>Options: Toolchain default, Enabled (-munaligned-access), Disabled (-mno-unaligned-access)</p> <p>Default: Toolchain default.</p> <p>Note: By default, unaligned access is enabled on all Arm® architectures, except for all pre-Arm®v6, all Arm®v6-M, and all Arm®v8-M.</p>
Libraries support	<p>The standard library and the I/O mode to be used for the application. Options:</p> <ul style="list-style-type: none"> • none: (Not recommended) Do not link the standard C/C++ library and disable support for console I/O. • newlib_nano no I/O: Link the lightweight NewLib and disable semihosting. • newlib_nano Debugger Console I/O: Link the lightweight NewLib and enable semihosting. • newlib no I/O: Link the standard NewLib with system C/C++ functions and disable semihosting. • newlib Debugger Console I/O: Link the standard NewLib with system C/C++ functions and use semihosting. • ewl_c no I/O: Link the standard Embedded Warrior Library (EWL) and disable semihosting. • ewl_c Debugger Console: Link the standard EWL and enable semihosting. • ewl_nano_c no I/O: Link the lightweight EWL and disable semihosting. • ewl_nano_c Debugger Console: Link the lightweight EWL and enable semihosting.
Sysroot	<p>The logical root location of headers and libraries.</p>

Standard S32DS C/C++ Compiler

This **Standard S32DS C/C++ Compiler** page displays the build configuration settings that apply to the set up the Standard S32DS C/C++ Compiler tool .

Table 39: Application and Library Project Properties: Standard S32DS C/C++ Compiler

Setting	Description
Command	<p>The command pattern for the $\\${COMMAND}$ variable. This variable is used in the Command line pattern field (below). Default patterns:</p> <ul style="list-style-type: none"> • C compiler: $\\${cross_prefix}\\${cross_c}\\${cross_suffix}$ • C++ compiler: $\\${cross_prefix}\\${cross_cpp}\\${cross_suffix}$ <p>The patterns use the build variables specified on the Cross Settings page.</p>
All options	<p>This read-only field aggregates all flags specified across all pages inside the compiler settings. The compiler will be called with these flags during the build process.</p>
Command line pattern	<p>The command line pattern to call the compiler.</p>

Setting	Description
	Default: $\${COMMAND} \ ${FLAGS} \ \${OUTPUT_FLAG}$ $\${OUTPUT_PREFIX}\${OUTPUT} \ \${INPUTS}$

Dialect

The **Dialect** page specifies the programming language standard and options to which the Standard S32DS C/C++ Compiler will conform.

Table 40: Application and Library Project Properties: Standard S32DS C/C++ Compiler > Dialect

Setting	Description
Language standard	The language standard to which the code should conform. The compiler accepts all programs that follow the specified standard plus GNU extensions that do not contradict it. Options: <ul style="list-style-type: none"> • C language: ISO C90/ANSI C89 (-std=c90), ISO C99 (-std=c99), ISO C11 (-std=c11), ISO C17 (-std=c17) • C++ language: ISO C++98 (-std=c++98), ISO C++11 (-std=c++11), ISO C++14 (-std=c++14), ISO C++17 (-std=c++17) Default: no option selected (the factory GCC standard applies). For more information, consult the GCC documentation at gcc.gnu.org .
Other dialect flags	Additional dialect options. You can specify any language options supported by GCC. Consult the GCC documentation.

Preprocessor

The **Preprocessor** page specifies the settings required by the GCC compiler for preprocessing source files.

Table 41: Application and Library Project Properties: Standard S32DS C/C++ Compiler > Preprocessor

Setting	Description
Do not search system directories (-nostdinc)	This option instructs the compiler to not search the system locations for header files. Only the locations specified on the Includes page will be searched.
Preprocess only (-E)	This option instructs the compiler to preprocess source files without doing the compilation step. Note: Selecting this option causes the linker to throw an error at build time. The linker expects an object file which is not created by the compiler.
Defined symbols (-D)	The prioritized list of symbols defined as macros. This option is similar to the #define directive but applies to all assembly-language modules in a build target.
Undefined symbols (-U)	The prioritized list of canceled symbols, both built-in and defined with the -D option.
Do not search system C++ directories (#nostdinc++)	This option instructs the C++ compiler to not search the system locations for header files. This option is only available for C++ projects.

Includes

The **Includes** page specifies header files to be used during compilation and the file paths to be searched for header files.

Table 42: Application and Library Project Properties: Standard S32DS C/C++ Compiler > Includes

Setting	Description
Include paths (-I)	The prioritized list of directories to be searched for header files. These directories are searched before the standard system include directories. Default paths: "\${ProjDirPath}/include"
Include files (-include)	The prioritized list of header files to be included.

Optimization

The **Optimization** page specifies optimizations run by the Standard S32DS C/C++ Compiler tool during the compilation of a program. Turning on optimization flags makes the C/C++ compiler attempt to improve the performance and code size at the expense of the compilation time and the ability to debug the program.

Table 43: Application and Library Project Properties: Standard S32DS C/C++ Compiler > Optimization

Setting	Description
Optimization level	The level of optimization assigned for the compiler. Options: <ul style="list-style-type: none"> • None (-O0): Disables optimization. This option instructs the compiler to generate unoptimized, linear assembly-language code. This reduces the compilation time. • Optimize (-O1): The compiler performs all target-independent (non-parallelized) optimizations such as function inlining, omits all target-specific optimizations, and generates linear assembly-language code. Optimizing takes somewhat more time, and a lot more memory for a large function. • Optimize more (-O2): The compiler performs all optimizations, target-independent and target-specific, and outputs optimized, non-linear, parallelized assembly-language code. This option increases both the compilation time and the performance of the generated code. • Optimize most (-O3): The compiler performs all -O2 optimizations, after which the low-level optimizer performs global-algorithm register allocation. At this optimization level, the compiler generates code that is usually faster than the code generated from -O2 optimizations. • Optimize size (-Os): The compiler performs further optimizations designed to reduce the code size. At this optimization level, the compiler performs all -O2 optimizations that do not typically increase the code size. The resulting binary file has a smaller executable code size, as opposed to a faster execution speed.
Other optimization flags	Additional optimization flags supported by GCC and not otherwise available on this page. Consult the GCC documentation at gcc.gnu.org .
'char' is signed (--fsigned-char)	This option instructs the compiler to treat char strings as signed char strings.
'bitfield' is unsigned (-funsigned-bitfields)	This option instructs the compiler to treat bit fields as unsigned.

Setting	Description
Function sections (-ffunction-sections)	This option instructs the compiler to place each function into a separate section in the binary artifact. Each function section will be given the name of the specific function placed in the section. Note: This option makes the assembler and linker create larger object and executable files and work slower.
Data sections (-fdata-sections)	This option instructs the compiler to place each data item into its own section in the output file. The name of the data item determines the section's name in the output file. Note: This option makes the assembler and linker create larger object and executable files and work slower.
No common uninitialized (-fno-common)	This option instructs the compiler to place uninitialized global variables in the data section of the object file rather than generating them as common blocks. This has the effect that if the same variable is declared (without extern) in two different compilations, you get a multiple-definition error when you link them. In this case, you must compile with <code>-fcommon</code> instead. Compiling with <code>-fno-common</code> is useful on targets for which it provides better performance, or if you wish to verify that the program will work on other systems that always treat uninitialized variable declarations this way.
Do not inline functions (-fno-inline-functions)	This option instructs the compiler to not consider any functions for inlining, even if they are not declared inline.
Assume freestanding environment (-ffreestanding)	This option instructs the compiler to assert that compilation targets a freestanding environment. This implies the use of the Disable builtin option (below). A freestanding environment is one in which the standard library may not exist, and the program startup may not necessarily be at <code>main</code> .
Disable builtin (-fno-builtin)	This option instructs the compiler to not recognize built-in functions that do not begin with the <code>__builtin_</code> as prefix.
Single precision constants (-fsingle-precision-constant)	This option instructs the compiler to treat floating-point constants as single-precision instead of implicitly converting them to double-precision constants.
Link-time optimizer (-flto)	This option instructs the compiler to run the standard link-time optimizer. When invoked with source code, it generates GIMPLE (one of GCC's internal representations) and writes it to special ELF sections in the object file. When the object files are linked together, all the function bodies are read from these ELF sections and instantiated as if they had been part of the same translation unit.
Disable loop invariant move (-fno-move-loop-invariants)	This option instructs the compiler to disable the loop invariant motion pass in the RTL loop optimizer. This option is available at optimization level “-O1”.

Debugging

The **Debugging** page specifies the debugging parameters to be used by the Standard S32DS C/C++ Compiler tool. The specified parameters affect the debugging information that will be available in the resulting build target: ELF executable file or A library file.

Table 44: Application and Library Project Properties: Standard S32DS C/C++ Compiler > Debugging

Setting	Description
Debug Level	The debugging level assigned to the compiler. Options: <ul style="list-style-type: none"> • None: Disables output of debugging information to the build artifact. • Minimal (-g1): Enables the generation of minimum debugging information. This includes descriptions of functions and external variables and line number tables, but no information about local variables. • Default (-g): Enables the generation of DWARF 1.x conforming debugging information. • Maximum (-g3): Enables the generation of extra debugging information (such as all macro definitions) for the compiler to provide maximum debugging support.
Other debugging flags	Additional debugging flags supported by GCC and not otherwise available on this page. Consult the GCC documentation at gcc.gnu.org .
Generate gcov information (-ftest-coverage -fprofile-arcs)	This option tells the compiler to generate additional information (basically a flow graph of the program) and to include additional code in the object files for generating the extra profiling information. These additional files are placed in the directory where the object file is located. The gcov code-coverage utility can use these additional files to test coverage of the program.
Debug format	The debug information format to be used by the compiler when writing debug info to the produced ELF file. Options: Toolchain default, gdb, stabs, stabs+, dwarf-2, dwarf-3, dwarf-4. Default: Toolchain default.

Warnings

The **Warnings** page specifies options used by the compiler to display warning messages during the compilation. These options specify what types of warning messages will be output in the console during the compilation.

Table 45: Application and Library Project Properties: Standard S32DS C/C++ Compiler > Warnings

Setting	Description
Check syntax only (-fsyntax-only)	This option instructs the compiler to only check the code for syntax errors.
Pedantic (-pedantic)	This option instructs the compiler to issue warnings when a program is rejected because of forbidden extensions or non-conformance to ISO C and ISO C++ standards. Not all non-ISO constructs get warnings. To learn more, consult the GCC documentation at gcc.gnu.org .
Pedantic warnings as errors (-pedantic-errors)	This option instructs the compiler to issue errors rather than warnings in cases described in the Pedantic option (above).
Inhibit all warnings (-w)	This option instructs the compiler to inhibit all warning messages.
All warnings (-Wall)	This options enables a group of GCC warning options on the compiler. This includes all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even

Setting	Description
	in conjunction with macros. This also enables some language-specific warnings. To learn more, consult the GCC documentation at gcc.gnu.org .
Extra warnings (-Wextra)	This option instructs the compiler to enable some extra GCC warning options that are not enabled by All warnings (above). For details, consult the GCC documentation at gcc.gnu.org .
Warnings as errors (-Werror)	This option instructs the compiler to turn all warnings into hard errors. The source code which triggers warnings will be rejected. The specifier for a warning is appended.
Implicit conversions warnings (-Wconversion)	This option instructs the compiler to issue warnings for implicit conversions that may alter a value. This includes conversions between real and integer, between signed and unsigned, and conversions to smaller types. No warning will be issued for explicit casts like <code>abs ((int)x)</code> and <code>ui=(unsigned)-1</code> , or if the value is not changed by the conversion.
Warn on uninitialized variables (-Wuninitialized)	This option instructs the compiler to issue a warning if an automatic variable is used without first being initialized or if a variable may be clobbered by a <code>set jmp</code> call.
Warn on various unused elements (-Wunused)	This option instructs the compiler to issue warnings if constructs are unused.
Warn if padding is included (-Wpadded)	This option instructs the compiler to issue a warning if padding is included in a structure, either to align an element of the structure or to align the whole structure.
Warn if floats are compared as equal (-Wfloat-equal)	This option instructs the compiler to issue warnings if floating-point values are used in equality comparisons.
Warn if shadowed variable (-Wshadow)	This option instructs the compiler to issue a warning whenever a local variable or type declaration shadows another variable, parameter, type, or class member (in C++), or whenever a built-in function is shadowed. Note: In C++, the compiler warns if a local variable shadows an explicit typedef, but not struct, or class, or enum.
Warn if pointer arithmetic (-Wpointer-arith)	This option instructs the compiler to issue a warning about anything that depends on the size of a function type or of void. GNU C assigns these types a size of 1, for convenience in calculations with <code>void *</code> pointers and pointers to functions.
Warn if suspicious logical ops (-Wlogical-op)	This option instructs the compiler to issue warnings about suspicious uses of logical operators in expressions. This includes using logical operators in contexts where a bit-wise operator is likely to be expected.
Warn if struct is returned (-Waggregate-return)	This option instructs the compiler to issue a warning if any functions that return structures or unions are defined or called.
Warn on undeclared global function (-Wmissing-declaration)	This option instructs the compiler to issue a warning if a global function is defined without a previous declaration. Use this option to detect global functions that are not declared in header files.

Setting	Description
Other warning flags	Additional command line options. Specify any options that control warning output in GCC and not otherwise available on this page. Consult the GCC documentation at gcc.gnu.org .

Miscellaneous

The **Miscellaneous** page specifies auxiliary compiler options not otherwise available on other pages of the Standard S32DS C/C++ Compiler settings.

Table 46: Application and Library Project Properties: Standard S32DS C/C++ Compiler > Miscellaneous

Setting	Description
Other flags	Additional command line options. Specify compiler options supported by GCC and not otherwise available on this page. Consult the GCC documentation at gcc.gnu.org . Default flags: <code>-c -fmessage-length=0</code> .
Verbose (-v)	This option enables verbose output during the compilation and instructs the compiler to display detailed information about the exact sequence of commands used to compile the program. Note: S32 Design Studio supports output of error messages only. Warning messages and other informational messages will not be output to the console.
Support ANSI programs (-ansi)	This option configures the compiler to operate in strict ANSI mode. This option is only available for the Standard S32DS C Compiler. This option turns off certain features of GCC that are incompatible with ISO C90 (when compiling C code), or of standard C++ (when compiling C++ code), such as the <code>asm</code> and <code>typeof</code> keywords, and predefined macros such as <code>unix</code> and <code>vax</code> that identify the type of system you are using. It also enables the undesirable and rarely used ISO trigraph feature. For the C compiler, it disables recognition of C++ style <code>/**</code> comments as well as the <code>inline</code> keyword. The macro <code>__STRICT_ANSI__</code> is predefined when this option is used. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ISO standard does not call for. Functions that are normally built in but do not have semantics defined by ISO C (such as <code>alloca</code> and <code>ffs</code>) are not built-in functions when this option is used.
Position Independent Code (-fpic)	This option instructs the compiler to generate position-independent code (PIC), if supported by the target processor.
Save temporary files (--save-temps)	This option instructs the compiler to save the result of preprocessing in a temporary I file and the result of assembling in a S file. The compiler places the files in the Debug folder of the project and names them based on the source file.
Generate assembler listing (-Wa, -adhlns="\$@.lst")	This option enables the compiler to output assembly listing to an LST file.
Assume aligned memory references only (-mstrict-align)	This option enables the compiler to access addresses not aligned to 16 or 32 bits. Note: This setting applies to projects created for Cortex-A53.

Standard S32DS C/C++ Linker

The **Standard S32DS C/C++ Linker** page displays the build configuration settings that apply to the C/C++ linker tool.

Table 47: Application Project Properties: Standard S32DS C/C++ Linker

Setting	Description
Command	<p>The command pattern for the <code>COMMAND</code> variable. This variable is used in the Command line pattern field (below). Default patterns:</p> <ul style="list-style-type: none"> C linker: <code>{cross_prefix}{cross_c}{cross_suffix}</code> C++ linker: <code>{cross_prefix}{cross_cpp}{cross_suffix}</code> <p>The pattern uses the build variables specified on the Cross Settings page.</p>
All options	This read-only field shows all flags specified on all pages of the C/C++ linker tool settings. The C/C++ linker will be called with these flags at build time.
Command line pattern	<p>The command line pattern to call the C/C++ linker tool.</p> <p>Default:</p> <pre>{COMMAND} {FLAGS} {OUTPUT_FLAG} {OUTPUT_PREFIX}{OUTPUT} {INPUTS}</pre>

General

The **General** page specifies the general properties of the Standard S32DS C/C++ Linker tool.

Table 48: Application Project Properties: Standard S32DS C/C++ Linker > General

Setting	Description
Do not use standard start files (-nostartfiles)	This option configures the linker to not use the standard system startup files when linking.
Do not use default libraries (-nodefaultlibs)	This option configures the linker to not use the standard system libraries (such as <code>newlib</code>) when linking. Only the customer-specified libraries can be passed to the linker.
No startup or default libs (-nostdlib)	This option configures the linker to not use the standard system startup files and libraries when linking. Only the customer-specified libraries can be passed to the linker.
Omit all symbols information (-s)	This option configures the linker to remove all symbol table and relocation table information from the executable.
No shared libraries (-static)	<p>This option prevents linking with the shared libraries. This option makes sense on systems that support dynamic linking.</p> <p>Note: The current version of S32 Design Studio supports linking with the static libraries only.</p>
Script files (-T)	The linker script.

Libraries

The **Libraries** page specifies custom libraries and their locations to be used by the Standard S32DS C/C++ Linker tool during compilation.

Table 49: Application and Library Project Properties: Standard S32DS Assembler > Libraries

Setting	Description
Libraries (-l)	The custom libraries to be linked to the application. The libraries will be linked in the top-down order they follow in the list.
Library search path (-L)	The file paths where the linker looks for custom libraries specified in the Libraries (-l) field. The linker searches the paths in the order they follow in the list.

Miscellaneous

The **Miscellaneous** page specifies additional linker-specific flags and options not directly related to linking.

Table 50: Application Project Properties: Standard S32DS C/C++ Linker > Miscellaneous

Settings	Description
Linker flags	Additional command line options (flags). You can specify any required options supported by the GNU linker and not otherwise available on this page. Consult the documentation at the GNU Binutils site.
Other options (-Xlinker [option])	System-specific linker options that GCC does not recognize. You can specify any options supported by the GNU linker and not otherwise available on this page. To specify options that take their arguments after the equal sign, specify the option and its argument as a single item. For example, to compress the debug section, add <code>--compress-debug-sections=zlib</code> . To specify options that take arguments after the space, add the option first, and then add the argument as an item that follows the option in the list. For example, to specify that <code>armelfd</code> emulation mode to be used, add <code>-m</code> as a single item, and then add <code>armelfd</code> as the item that follows.
Other objects	The prioritized list of object file paths to be used when linking. The added paths are stores in the <code>{application}.args</code> file located in the Debug folder of the project. If you add a relative path that starts with a period, put the path string in quotes.
Generate map	The name of the MAP file to be generated by the linker. The generated MAP file lists the resource data in the ELF file. The MAP file is located in the Debug folder of the project structure. To instruct the linker to not generate the MAP file but print the resource information to the console, leave this setting blank. Default: <code>\$ {BuildArtifactFileName}.map</code>
Cross reference (-Xlinker --cref)	This option instructs the linker to print the cross reference table. If the Generate map setting (above) specifies the file name, the table is printed to the specified MAP file. Otherwise, the table is printed to the console. This cross reference table includes the “symbol - files” pairs. The symbols are sorted in alphabetical order, each followed by the file paths where this symbol is defined.

Settings	Description
Print link map (-Xlinker --printf-map)	This option instructs the linker to print the resource map to the console if the Generate map setting (above) is blank. If the Generate map setting specifies the MAP file name, the linker ignores this option and prints the resource information to the MAP file.
Remove unused sections (-Xlinker --gc-sections)	This option removes unused sections of code and data from the binary artifact.
Print removed sections (-Xlinker --print-gc-sections)	This option instructs the linker to print the removed sections to the Console view. This option makes sense if the Remove unused sections option (above) is enabled.
Support print float format for newlib_nano library (-u_printf_float)	This option enables printing of floating-point formatted numbers to the console. This option assumes that semihosting is enabled and the system write function is provided by the debugger. This option is grayed out by default. To make it available, set Library support to newlib_nano Debugger Console on the Target processor page. When enabled, this option increases the heap and stack size.
Support scan float format for newlib_nano library (-u_printf_float)	This option enables scanning of floating-point formatted numbers from the console. This option assumes that semihosting is enabled and the system read function is provided by the debugger. This option is grayed out by default. To make it available, set Library support to newlib_nano Debugger Console on the Target processor page. When enabled, this option increases the heap and stack size.
EWL print formats	Not used in the current version of the product. This option specifies the print format for numbers. Options: none, int, int_FP, int_LL_FP. Default: none. This option is grayed out by default. To make it available, set Library support to ewl_nano_c Debugger Console/newlib_nano_c++ Debugger Console on the Target processor page.
EWL scan formats	Not used in the current version of the product. This option specifies the scan format for numbers. Options: none, int, int_FP, int_LL_FP. Default: none. This option is grayed out by default. To make it available, set Library support to ewl_nano_c Debugger Console/newlib_nano_c++ Debugger Console on the Target processor page.

Shared Library Settings

The properties on the **Shared Library Settings** page are not supported by the current version of S32 Design Studio.

Link Order

The **Link Order** page specifies the order in which input files are passed to the linker.

Table 51: Application Project Properties: Standard S32DS C/C++ Linker > Link Order

Setting	Description
Customize linker input order	This option enables you to reorder files in the list (below).
Link Order	The prioritized list of files that are passed to the linker as inputs.

Standard S32DS Assembler

The **Standard S32DS Assembler** page displays the build configuration settings that apply to the GNU assembler tool.

Table 52: Application and Library Project Properties: Standard S32DS Assembler

Setting	Description
Command	The command pattern for the <code>\$(COMMAND)</code> variable. This variable is used in the Command line pattern field (below). Default: <code>\$(cross_prefix)\$(cross_c)\$(cross_suffix)</code> The pattern uses the build variables specified on the Cross Settings page.
All options	This read-only field shows all flags specified on all pages of the assembler tool settings. The assembler will be called with these flags at build time.
Command line pattern	The command line pattern to call the assembler tool. Default: <code>\$(COMMAND) \$(FLAGS) \$(OUTPUT_FLAG)</code> <code>\$(OUTPUT_PREFIX)\$(OUTPUT) \$(INPUTS)</code>

General

The **General** page specifies the general properties used by the Standard S32DS assembler tool.

Table 53: Application and Library Project Properties: Standard S32DS Assembler > General

Setting	Description
Assembler flags	Additional command line options supported by the GNU assembler and not otherwise available on this page. Consult the GNU documentation at GNU Binutils . Default flags: <code>-c</code> (runs the assembler without linking).
Include paths (-I)	The prioritized list of paths for include files lookup. Default: <code>"\$(ProjDirPath)/include"</code> for applications; no path for libraries.
Suppress warnings (-W)	This option suppresses output of assembler-generated warning messages to the console.
Announce version (-v)	This option enables the assembler tool to show extended information about the assembly progress, including the GCC version, variables being used, and informational messages returned while assembling the code.

Preprocessor

The **Preprocessor** page configures the assembly preprocessor that is run on each S file before assembling it.

Table 54: Application and Library Project Properties: Standard S32DS Assembler > Preprocessor

Setting	Description
Use preprocessor	This option enables the preprocessor.
Do not search system directories (-nostdinc)	This option configures the preprocessor to not search the system locations for header files. Only the locations specified on the General page will be searched.
Preprocess only (-E)	This option tells the preprocessor to handle source files and stop. The compiler will not be run. Note: Enabling this option will cause the linker to throw an error at build time. This happens because the linker expects an object file which is not created because no compilation is done.

Symbols

The **Symbols** page defines the assembly symbols for the Standard S32DS Assembler tool.

Table 55: Application and Library Project Properties: Standard S32DS Assembler > Symbols

Setting	Description
Defined symbols (-D)	The prioritized list of substitution strings that the assembler applies to all assembly-language modules in the build target. Note: The -D token is added automatically to each string that you enter. For example, entering “opt1 x” results in the “-Dopt1 x” list entry.
Undefined symbols (-U)	The list of the built-in assembler symbols to be suppressed.

Debugging

The **Debugging** page specifies the debugging options used by the Standard S32DS Assembler tool.

Table 56: Application and Library Project Properties: Standard S32DS Assembler > Debugging

Setting	Description
Debug Level	The debugging level assigned to the assembler. Options: <ul style="list-style-type: none"> • None: Disables output of debugging information to the build artifact. • Minimal (-g1): Enables the generation of minimum debugging information. This includes descriptions of functions and external variables and line number tables, but no information about local variables. • Default (-g): Enables the generation of DWARF 1.x conforming debugging information. • Maximum (-g3): Enables the generation of extra debugging information for the compiler to provide maximum debugging support.
Other debugging flags	Additional debugging flags supported by the GNU assembler and not otherwise available on this page.

Standard S32DS Archiver

The **Standard S32DS Archiver** page displays the build configuration settings that apply to the archiver tool. This page is only available for library projects.

Table 57: Library Project Properties: Standard S32DS Archiver

Setting	Description
Command	The command pattern for the <code>COMMAND</code> variable. This variable is used in the Command line pattern field (below). Default: <code> \${cross_prefix} \${cross_ar} \${cross_suffix}</code> The pattern uses the build variables specified on the Cross Settings page.
All options	This read-only field shows all flags specified on the General page of the archiver tool settings. The archiver tool will be called with these flags. Default flags: <code>-r</code>
Command line pattern	The command line pattern to call the archiver tool. Default: <code> \${COMMAND} \${FLAGS} \${OUTPUT_FLAG}</code> <code> \${OUTPUT_PREFIX} \${OUTPUT} \${INPUTS}</code>

General

The **General** page specifies the general properties used by the Standard S32DS Archiver virtual tool.

Table 58: Library Project Properties: Standard S32DS Archiver > General

Setting	Description
Archiver flags	Additional archiver tool options not included in the project properties. You can specify any required options supported by the archiver tool. Consult the GCC documentation at gcc.gnu.org . Default: <code>-r</code> (add and replace).

Standard S32DS Create Flash Image

The **Standard S32DS Create Flash Image** page displays the build configuration settings that apply to the Standard S32DS Create Flash Image virtual tool. This tool calls the HEX/BIN converter tool to create a flash image from an application's executable.

This page is only available for application projects. To make this page available, enable the **Create flash image** option on the Cross Settings page.

Table 59: Application Project Properties: Standard S32DS Create Flash Image

Setting	Description
Command	The command pattern for the <code>COMMAND</code> variable. This variable is used in the Command line pattern field (below). Default: <code> \${cross_prefix} \${cross_objcopy} \${cross_suffix}</code> The pattern uses the build variables specified on the Cross Settings page.
All options	This read-only field shows all flags specified on the General page of the Standard S32DS Create Flash Image tool settings. The HEX/BIN converter tool will be called with these flags. Default flags: <code>-O srec \$(EXECUTABLES)</code>

Setting	Description
Command line pattern	The command line pattern to run the HEX/BIN converter tool. Default: <code>\${COMMAND} \${FLAGS} \${OUTPUT_FLAG} \${OUTPUT_PREFIX} \${OUTPUT}</code>

General

The **General** page specifies the general properties of the Standard S32DS Create Flash Image virtual tool. This tool calls the HEX/BIN converter tool to generate an image file in the selected binary format from the produced binary ELF artifact.

Table 60: Application Project Properties: Standard S32DS Create Flash Image > General

Setting	Description
Output file format (#O)	The binary format of the flash image. Options: Intel HEX, Motorola S-record, Motorola S-record (symbols), RAW binary. The generated image file is added to the Debug folder of the project. The file name matches the project name, the file extension depends on the selected format and can be <code>.hex</code> (Intel HEX), <code>.srec</code> (Motorola S-record), <code>.symbolsrec</code> or <code>.bin</code> . Default: Motorola S-record.
Section: -j .text	This option configures the HEX/BIN converter tool to include only the TEXT section of the ELF file into the flash image. Other file sections will not be included unless specified explicitly in the Other sections (-j) list.
Section: -j .data	This option configures the HEX/BIN converter tool to include only the DATA section of the ELF file into the flash image. Other file sections will not be included unless specified explicitly in the Other sections (-j) list.
Other sections (-j)	Specify other sections of the ELF file to be included in the flash image file. Note: To add both the TEXT section and the DATA section to the image, select any of the respective options above and add the other file section here.
Other flags	Additional flags supported by the HEX/BIN converter tool and not otherwise available on this page. Consult the GCC documentation at gcc.gnu.org . To specify additional flags, use the following pattern: <code>--set-section-flags sectionpattern=flags</code> Example: <code>--set-section-flags .text=alloc</code>

Standard S32DS Create Listing

The **Standard S32DS Create Listing** page displays the build configuration settings that apply to the Standard S32DS Create Listing virtual tool. This tool calls the listing generator tool to disassemble the application's ELF binary and to generate the disassembly listing for the build target of the current project. The generated `.lst` file with the project name can be found in the Debug folder of the project.

This page is only available for application projects. To make this page available, enable the **Create extended listing** option on the Cross Settings page.

Table 61: Application Project Properties: Standard S32DS Create Listing

Setting	Description
Command	The command pattern for the <code>{COMMAND}</code> variable. This variable is used in the Command line pattern field (below). Default: <code>{cross_prefix}{cross_objdump}{cross_suffix}</code> The pattern uses the build variables specified on the Cross Settings page.
All options	This read-only field shows all flags specified on the General page of the Standard S32DS Create Listing tool settings. The listing generator tool will be called with these flags. Default flags: <code>--source --all-headers --demangle --line-numbers --wide \$(EXECUTABLES)</code>
Command line pattern	The command line pattern to run the listing generator tool. Default: <code>{COMMAND} {FLAGS} {OUTPUT_FLAG} {OUTPUT_PREFIX}{OUTPUT}</code>

General

The **General** page configures the general properties of the Standard S32DS Create Listing virtual tool. The tool calls the listing generator tool to generate the LST file with a disassembly of the produced binary artifact.

The following flags configure the listing generator tool to include particular information in the generated disassembly file.

Table 62: Application Project Properties: Standard S32DS Create Listing > General

Setting	Description
Display source (--source -S)	Includes the source code intermixed with disassembly.
Display all headers (--all-headers -x)	Shows all available header information, including the symbol table and relocation entries.
Demangle names (--demangle -C)	Makes the disassembled function names more user-friendly and legible by decoding mangling styles used by the compiler.
Display debugging info (--debugging -g)	Includes debugging information obtained from the artifact file. Note: For the application's ELF file to contain debugging information, configure the compiler to use the STABS or DWARF format. You can do it on the Debugging page of the Standard S32DS C/C++ Compiler tool settings.
Disassemble (--disassemble -d)	Includes assembler mnemonics for the machine instructions from the object file. The tool disassembles only those sections in the ELF file that are expected to contain instructions.
Display file headers (--file-headers -f)	Includes summary information from the overall header of each of the object files.
Display line numbers (--line-numbers -l)	Includes the file name and path, the line numbers corresponding to the object code or relocation entries. This option requires the Disassemble (--disassemble -d) or Display relocation info (--reloc -r) option to be enabled.

Setting	Description
Display relocation info (--reloc -r)	Includes entries from the relocation table. Note: To add disassembly information to the output relocation data, enable the Disassemble (--disassemble -d) option.
Display symbols (--syms -t)	Includes entries from the symbol table.
Wide lines (--wide -w)	Formats lines for output devices that have more than 80 columns. Use this option to avoid truncation of symbol names.
Other flags	Additional options supported by the listing generator (objdump) tool in GCC and not otherwise available on this page. Consult the GCC documentation at gcc.gnu.org .

Standard S32DS Print Size

The **Standard S32DS Print Size** page displays the build configuration settings that apply to the size tool. This tool prints the size of the produced application.

This page is only available for application projects.

Table 63: Application Project Properties: Standard S32DS Print Size

Setting	Description
Command	The command pattern for the <code>\${COMMAND}</code> variable. This variable is used in the Command line pattern field (below). Default: <code>\${cross_prefix}\${cross_size}\${cross_suffix}</code> The pattern uses the build variables specified on the Cross Settings page.
All options	This read-only field shows all flags specified on the General page of the size tool settings. The tool will be called with these flags. Default flags: <code>--format=berkeley \$(EXECUTABLES)</code>
Command line pattern	The command line pattern to run the size tool. Default: <code>\${COMMAND} \${FLAGS}</code>

General

The **General** page configures the general properties of the size tool. The tool prints the size of the built application.

Table 64: Application Project Properties: Standard S32DS Print Size > General

Setting	Description
Size format	The output format. Options: Berkeley, SysV. Default: Berkeley.
Hex	Enables the tool to show the size of each section in hexadecimal format. Default: disabled (decimal format is used).
Show totals	Enables the tool to show totals of all objects listed. This option applies to the Berkeley output format only.

Setting	Description
Other flags	Additional flags supported by the size tool in GCC and not otherwise available on this page. Consult the GCC documentation at gcc.gnu.org .

Standard S32DS C/C++ Preprocessor

The **Standard S32DS C/C++ Preprocessor** page describes the build configuration settings that apply to the preprocessor tool.

Note: When you build a project, this tool is not called. Use it at any time as a standalone tool to preprocess the selected source files and to preview the output. Find the details in topic [Preprocessing source files](#).

Table 65: Application Project and Library Properties: Standard S32DS C/C++ Preprocessor

Setting	Description
Command	The command pattern for the <code>\$(COMMAND)</code> variable. This variable is used in the Command line pattern field (below). Default patterns: <ul style="list-style-type: none"> C preprocessor: <code>\$(cross_prefix)\$(cross_c)\$(cross_suffix)</code> C++ preprocessor: <code>\$(cross_prefix)\$(cross_cpp)\$(cross_suffix)</code> The pattern uses the build variables specified on the Cross Settings page.
All options	This read-only field shows all flags specified on the Settings page of the Standard S32DS C/C++ preprocessor tool settings. The preprocessor will be called with these flags. Default flags: <code>-E</code>
Command line pattern	The command line pattern to call the preprocessor tool. Default: <code>\$(COMMAND) \$(FLAGS) \$(INPUTS)</code>

Settings

The **Settings** page configures the general properties of the Standard S32DS C/C++ Preprocessor virtual tools. These tools perform preprocessing of the selected C or C++ source files without building the project.

Table 66: Application Project and Library Properties: Standard S32DS C/C++ Preprocessor > Settings

Setting	Description
Handle Directives Only (fdirectives-only)	Enables preprocessing of directives such as <code>#define</code> , <code>#ifdef</code> , and <code>#error</code> without expanding macros. Note: The <code>-E</code> flag automatically enabled in the command line of the preprocessor limits preprocessing to handling only the compiler directives. Expansion of macros and conversion of trigraphs are not performed.
Print Header File Names (-H)	Enables the preprocessor to scan include directories and output name of every header file included in your code. The output is redirected to the console.

Standard S32DS Disassembler

The **Standard S32DS Disassembler** page describes the build configuration settings that apply to the disassembler tool. This tool calls the listing generator tool to generate disassembly of a binary or source file.

Note: When you build a project, the disassembler tool is not called. Use it at any time as a standalone tool to generate the disassembly listings for the selected binary and source files and to preview the output. Find the details in topic [Disassembling source files](#).

Table 67: Application Project and Library Properties: Standard S32DS Disassembler

Setting	Description
Command	The command pattern for the <code> \${COMMAND} </code> variable. This variable is used in the Command line pattern field (below). Default: <code> \${cross_prefix} \${cross_objdump} \${cross_suffix} </code> The pattern uses the build variables specified on the Cross Settings page.
All options	This read-only field shows all flags specified for the Standard S32DS Disassembler tool on the Settings page. The disassembler will be called with these flags. Default flags: <code> -d -S -x </code>
Command line pattern	The command line pattern to call the listing generator tool. Default: <code> \${COMMAND} \${FLAGS} \${INPUTS} </code>

Settings

The **Settings** page configures the general properties of the Standard S32DS Disassembler virtual tool. This tool generates and outputs the disassembly listing for any selected binary and C/C++ source files without building the project.

The following flags configure the tool to include particular information in the generated disassembly output.

Table 68: Application Project and Library Properties: Standard S32DS Disassembler > Settings

Setting	Description
Disassemble All Section Content (including debug information) (-D)	This option configures the tool to disassemble across all sections of the file. The tool decodes pieces of data found in code sections as if they were instructions.
Disassemble Executable Section Content (-d)	This option configures the tool to show the assembler mnemonics for machine instructions in sections that are known to contain instructions. Other sections are skipped.
Intermix Source Code With Disassembly (-S)	This option configures the tool to add the source code to the disassembled code where possible.
Display All Header Content (-x)	This option configures the tool to show full information about headers, including symbol table and relocation entries.
Display Archive Header Information (-a)	This option configures the tool to extract information from the header of the A archive that wraps the O file of your library. Additionally, this option enables the tool to display the format of O files contained within the archive. This option can be configured when option Display All Header Content (-x) is not selected (see above).
Display Overall File Header Content (-f)	This option configures the tool to extract information from the overall header of the file.

Setting	Description
	This option can be configured when option Display All Header Content (-x) is not selected (see above).
Display Object Format Specific File Header Contents (-p)	<p>This option configures the tool to output information specific to the header format of the file being disassembled. Availability of this information depends on the file format. Some formats may not contain this information, and no detail is output in the disassembly.</p> <p>This option can be configured when option Display All Header Content (-x) is not selected (see above).</p>
Display Section Header Content (-h)	<p>This option configures the tool to show file sections such as .TEXT, .DATA in the disassembly.</p> <p>This option can be configured when option Display All Header Content (-x) is not selected (see above).</p>
Display Full Section Content (-s)	This option configures the tool to show all data contained within file sections, including zero data in empty sections.
Display Debug Information (-g)	<p>This option configures the tool to obtain debugging information stored in the artifact file and print it out using a C-like syntax.</p> <p>The tool parses STABS and IEEE debugging format information stored in the produced file. If neither of these formats are found in the ELF file, the tool will attempt to print DWARF information available in the file.</p> <p>Note: If you want the ELF file of your application to contain debugging information, make sure to configure the compiler to use STABS or DWARF formats. You can specify the format on the Debugging page under of the Standard S32DS C/C++ Compiler settings.</p>
Display Debug Information Using ctag Style (-e)	This option configures the tool to display information about the TAG file created by the ctags tool. Output may contain information from the disassembler tool itself.
Display STABS Information (-G)	This option configures the tool to display debugging contents of the .STABS section in the file being disassembled.
Display DWARF Information (-W)	This option configures the tool to display debugging contents of debug sections such as .DEBUG_INFO, .DEBUG_FRAME, if present in the file being disassembled.
Display Symbol Table Content (-t)	This option configures the tool to show entries from the symbol table.
Display Dynamic Symbol Table Content (-T)	This option configures the tool to show entries from the dynamic table of symbols (.DYNSYM) that are added to the ELF file at runtime.
Display Relocation Entries (-r)	This option configures the tool to show entries from the relocation table created by assembler.
Display Dynamic Relocation Entries (-R)	This option configures the tool to show entries from the dynamic relocation table created by assembler.

Folders and files

Project structure

The following table describes the standard set of folders and files generated for an application project and displayed in the **Project Explorer** view. All folders and files are located inside the project's root folder.

Table 69: S32DS Application Project: Folders and files

Folder/file	Description
Binaries	This virtual folder appears after the project build and references the generated executable file (<project_name>.elf).
Includes	This virtual source folder contains the list of all discovered header files, including the header files used in the project directly.
Project_Settings	This folder includes the lower-level folders: <ul style="list-style-type: none"> Startup_Code: Includes the initialization scripts generated by the project creation wizard. Debugger: Includes the launch configurations (LAUNCH files) generated by the wizard. Linker_Files: Includes the linker files generated by the wizard.
SDK	(Optional) This folder is available if the SDK is attached to project. The SDK descriptor specifies which files will be copied to this project folder.
board	(Optional) This folder is available if the project uses the device configuration feature. The folder has no content when created. When the user configures MCU pins and clocks, the source files with code are generated and placed in this folder.
include	This folder includes the toolchain header files.
src	This folder includes the source files. The main.c or main.cpp file is included by default, other files can be added by the user.
<build_configuration_name>	This folder appears after the project build. The name of the folder matches the name of the build configuration used for the build (Debug or Release, or a custom configuration). The following lower-level folders are generated inside: <ul style="list-style-type: none"> board: Includes the O, D, ARG files and the makefile generated from the source files located in the <project_root_folder>/board folder. Project_Settings/Startup_Code: Includes the O, D, ARG files and the makefile generated from the initialization scripts located in the <project_root_folder>/Project_Settings/Startup_Code folder. SDK: Includes the O, D, ARG files and the makefile generated from the source files located in the <project_root_folder>/SDK folder.

Folder/file	Description
	<ul style="list-style-type: none"> src: Includes the O, D, ARG files and the makefile generated from the source files located in the <project_root_folder>/src folder. <p>The folder also includes files <project_name>.elf (executable), <project_name>.arg, <project_name>.map and makefiles generated for the project.</p>
description.txt	This file includes a brief description of the project. The text is entered by the user in the project creation wizard.
<processor_family>.mex	(Optional) This file stores the device configuration in the XML format. The file is available if the project supports the device configuration feature.

The following table describes the standard set of folders and files generated for a library project and displayed in the **Project Explorer** view. All folders and files are located inside the project's root folder.

Table 70: S32DS Library Project: Folders and files

Folder/file	Description
Archives	This virtual folder appears after the project build and references the generated archive file (lib<project_name>.a) with the project's object file inside.
Includes	This virtual source folder contains the list of all discovered header files, including the header files used in the project directly.
SDK	(Optional) This folder is available if the SDK is attached to project. The SDK descriptor specifies which files will be copied to this project folder.
include	This folder includes the toolchain header files.
src	This folder includes the source files. The mylibrary.c or mylibrary.cpp file is included by default, other files can be added by the user.
<build_configuration_name>	<p>This folder appears after the project build. The name of the folder matches the name of the build configuration used for the build (Debug or Release, or a custom configuration).</p> <p>The following lower-level folders are generated inside:</p> <ul style="list-style-type: none"> board: Includes the O, D, ARG files and the makefile generated from the source files located in the <project_root_folder>/board folder. SDK: Includes the O, D, ARG files and the makefile generated from the source files located in the <project_root_folder>/SDK folder. src: Includes the O, D, ARG files and the makefile generated from the source files located in the <project_root_folder>/src folder. <p>The folder also includes the lib<project_name>.a archive file (containing the project's object file) and makefiles generated for the project.</p>

Product directory structure

The following table describes the standard set of folders and files located inside the product installation directory.

Table 71: S32 Design Studio: folders and files

Folder/file	Description
_S32 Design Studio for S32 Platform 3.4_installation	This folder contains the following: <ul style="list-style-type: none"> the installation log the uninstaller other files related to the installation and uninstallation processes
Drivers	This folder contains the installation wizard of P&E Device Drivers.
eclipse	This folder contains eclipse features, plugins, configuration files, etc.
jre	This folder contains Java Runtime Environment which provides complete runtime support. This feature is included in the installation package for Windows only. For Linux platform, JRE must be installed separately.
Release_Notes	This folder contains product related Release Notes.
S32DS	This folder includes all tools and resources specific for S32 Design Studio for S32 Platform. The following lower-level folders can be found inside: <ul style="list-style-type: none"> <code>build_tools</code>: compiler and assembler tools necessary to build various types of projects <code>cll</code>: license components <code>config</code>: configuration and extension files <code>examples</code>: example projects added by user <code>help</code>: product documentation, including user guides, hardware manuals, tutorials and the 'Getting Started' video <code>integration</code>: software manifest files <code>software</code>: integrated SDKs and libraries <code>tools</code>: debugger, flash, software analysis tools; accelerator specific tools (contributed with the respective package), gdb
SCR	This folder contains the Software Content Register files.
s32ds.bat/s32ds.sh	This file sets environment variables and starts S32 Design Studio for S32 Platform.
Welcome.txt	This file includes a brief introduction.

How to Reach Us:

Home Page:

nxp.com

Web Support:

nxp.com/support

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

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP 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 NXP 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. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP Semiconductors has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP Semiconductors accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Airfast, Altivec, CodeWarrior, ColdFire, ColdFire+, CoolFlux, CoolFluxDSP, the CoolFlux logo, EdgeLock, EdgeScale, EdgeVerse, eIQ, Embrace, Freescale, the Freescale logo, GreenChip, the GreenChip logo, HITAG, ICODE, I - CODE, Immersiv3D, JCOP, Kinetis, Layerscape, MagniV, Mantis, MIFARE, the MIFARE logo, MIFARE CLASSIC, MIFARE DESFire, MIFARE FleX, MIFARE Plus, MIFARE Ultralight, MIFARE 4Mobile, the MIFARE4Mobile logo, MiGLO, mobileGT, NTAG, the NTAG logo, PEG, Plus X, PowerQUICC, Processor Expert, QoriQ, QoriQ Qonverge, Qorivva, RoadLINK, the RoadLINK logo, SafeAss ure, SmartM X, StarCore, Symphony, Tower, TriMedia, UCODE, the UCODE DNA logo, VortiQa and Vybrid are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μ Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2017-2021

All rights reserved.

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: salesaddresses@nxp.com

Revision: 2.0, April 2021

