S32 Design Studio for Power Architecture, Version 2017.R1

Reference Manual

Contents

Chapter 1

Introduction

Topics:

This manual explains how to use the S32 Design Studio for Power Architecture, Version 2017.R1 (S32DS Power v2017.R1) product for easier application creation and configuration with graphical user interface. This chapter presents an overview of the manual and introduces you to the information layout of the manual.

The topics in this chapter are:

Quick links	Lists web-links to S32 Design Studio pages
Terminology	Lists terms used in the document
About this manual	Describes the contents of this manual
Accompanying documentation	Describes supplementary S32DS Power v2017.R1 documentation, third-party documentation, and references
Manual conventions	Lists conventions used in the document
S32DS Power v2017.R1 content	Describes S32DS Power v2017.R1 contents
Installing S32DS Power v2017.R1	Describes the installation of S32DS Power v2017.R1 software
Uninstalling S32DS Power v2017.R1	Describes the uninstallation of S32DS Power v2017.R1

Quick links

- S32 Design Studio page (overview, downloads) — www.nxp.com/S32DS
- S32 Design Studio community (for publicly shared cases) — community.nxp.com/community/s32/s32ds
- Technical support (for confidential issues) — www.nxp.com/support Hardware and Software link

Terminology

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

Table 1: Terminology

Term	Description
ARM	Acorn RISC Machine architecture - family of instruction set architectures for computer processors developed by British company ARM Holdings, based on a reduced instruction set computing (RISC) architecture
CDT	C/C++ Development Tooling - a fully functional C and C++ IDE based on the Eclipse platform
DWARF	Debugging With Attributed Record Formats - a widely used, standardized debugging data format
ELF	Executable and Linkable Format - a common standard file format for executables, object code, shared libraries, and core dumps
EmbSys Registers	EMBedded SYStems REGister View - an Eclipse Plugin designed for monitoring and modifying memory values of embedded devices
FPU	Floating-point unit - math coprocessor; a part of a computer system specially designed to carry out operations on floating point numbers
IDE	Integrated development environment
GCC	GNU Compiler Collection - a compiler system produced by the GNU Project supporting various programming languages
GDB	GNU Debugger
KDS	Kinetis Design Studio - a complimentary integrated development environment for Kinetis MCUs that enables robust editing, compiling and debugging of your designs
P&E	P&E Microcomputer Systems
RAM	Random-access memory
S32DS Power v2017.R1	S32 Design Studio for Power Architecture, Version 2017.R1
XML	Extensible Markup Language - a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable

About this manual

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

Table 2: Manual Contents

Chapter	Description
Introduction	This chapter presents an overview of the manual and introduces you to the information layout of the manual
Working with projects	Explains how to use the S32DS Power v2017.R1 tools to create and work with projects
Build properties for S32DS Power v2017.R1 Projects	Explains build properties for a S32DS Power v2017.R1 project
Working with debugger	Explains how to use the S32DS Power v2017.R1 development tools to debug a program executing on the microcontroller
Multicore debugging	Explains how to define multiple, arbitrary groupings of cores and perform multicore operations
Connections	Describes the features and settings of the connections that interface the debugger with bareboard target and allows it to debug program code on the target
SDKs	Describes usage of the software development kits (SDKs) in the S32DS Power v2017.R1

Accompanying documentation

S32DS Power v2017.R1 includes an extensive documentation library of user guides, reference manuals etc. Take advantage of this library to learn how to efficiently develop software using the S32DS Power v2017.R1 programming environment.

S32DS Power v2017.R1 documentation presents information in the following formats:

- **PDF** Portable Document Format of the manuals, such as the Common Features Guide, Reference Manual or Release Notes
- **HTML** Hypertext Markup Language version of the manuals

PDF-documentation

The **S32 Design Studio Documentation Suite** includes the S32DS Power v2017.R1 PDF-documentation.

You can access the manuals, FAQ, etc. by:

- opening the **start_here.html** in **<S32 Design Studio install dir>/S32DS/help** directory, where **S32 Design Studio Install dir** is the directory that S32 Design Studio was installed into
- selecting **Help** > **Documentation** from the S32DS Power v2017.R1 menu bar
- selecting the **Documentation** shortcut.

Online help

To view the online help:

1. Select **Help** > **Help Contents** from the S32DS Power v2017.R1 menu bar.
2. Select required manual from the **Contents** list.

Release notes

Before using the S32DS Power v2017.R1, read the developer notes. These notes contain important information about last-minute changes, late-breaking information about new features, bug fixes, incompatible elements, known problems, or other topics that may not be included in this manual.

Read the Release notes in this directory:

<S32 Design Studio install dir>/S32DS/Release_Notes.

The release notes for specific components of the S32DS Power v2017.R1 are located in the **Release_Notes** directory in the S32DS Power v2017.R1 installation directory.

Manual conventions

This manual contains illustrations of user interface. Some S32DS Power v2017.R1 interface elements can differ from described in the manual.

You can use keyboard shortcuts, or key bindings, for frequently used operations in the S32DS Power v2017.R1 IDE. The **Menus and Keyboard Shortcuts** lists key bindings for frequently used operations. You can obtain a list of key bindings using **Help** > **Show Active Keybindings...** or **Ctrl+Shift+L**. Refer to the **Getting Started** > **Quick Reference Card** > **Menus and Keyboard Shortcuts** topic for details.

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

S32DS Power v2017.R1 content

S32DS Power v2017.R1 includes following features and tools:

- Eclipse Luna 4.4 framework
- GNU Build Tools for e200 processors (support VLE and BookE ISA, based on gcc 4.9.4, binutils 2.28 and gdb 7.8.2)
- Libraries included: newlib, newlib-nano and Freescale EWL2
- PnE Multilink/Cyclone (with PnE GDB Server)
- New S32DS Project wizard to create application and library projects for supported devices
- Peripherals Register view and Special Purpose Registers view
- Graphical tool for creating SPT algorithms
- SDK management included:
  - FreeMaster Serial Communication driver 2.0
  - Automotive Math and Motor Control Libraries
- GreenHills and Diab compilers support by the New S32DS Project wizard
- Lauterbach, PLS and iSystem debuggers support by the New S32DS Project wizard
- Kernel Aware debugging for FreeRTOS, eCOS, OSEK
- Examples included:
  - Single Elf example of multicore project
  - Example with Bootloader for MPC5748G device
- Command “New S32DS project from” to support project creating from examples
- Support of the next devices:
  - S32R372, S32R274

- MPC5601D, MPC5601P
- MPC5602B, MPC5602C, MPC5602D, MPC5602P
- MPC5603B, MPC5603C, MPC5603P
- MPC5604B, MPC5604C, MPC5604E, MPC5604P
- MPC5605B
- MPC5606B, MPC5606S
- MPC5607B
- MPC5632M, MPC5633M, MPC5634M
- MPC5642A
- MPC5643L
- MPC5644A, MPC5644B, MPC5644C
- MPC5645B, MPC5645C, MPC5646B, MPC5646C
- MPC5673K
- MPC5674F, MPC5674K
- MPC5675K
- MPC5676R
- MPC5744P, MPC5743P, MPC5742P, MPC5741P
- MPC5746R, MPC5745R, MPC5743R
- MPC5748G, MPC5747G, MPC5746G
- MPC5745B, MPC5744B, MPC5746B, MPC5744C, MPC5745C, MPC5746C
- MPC5774K, MPC5775K
- MPC5777C, MPC5777M

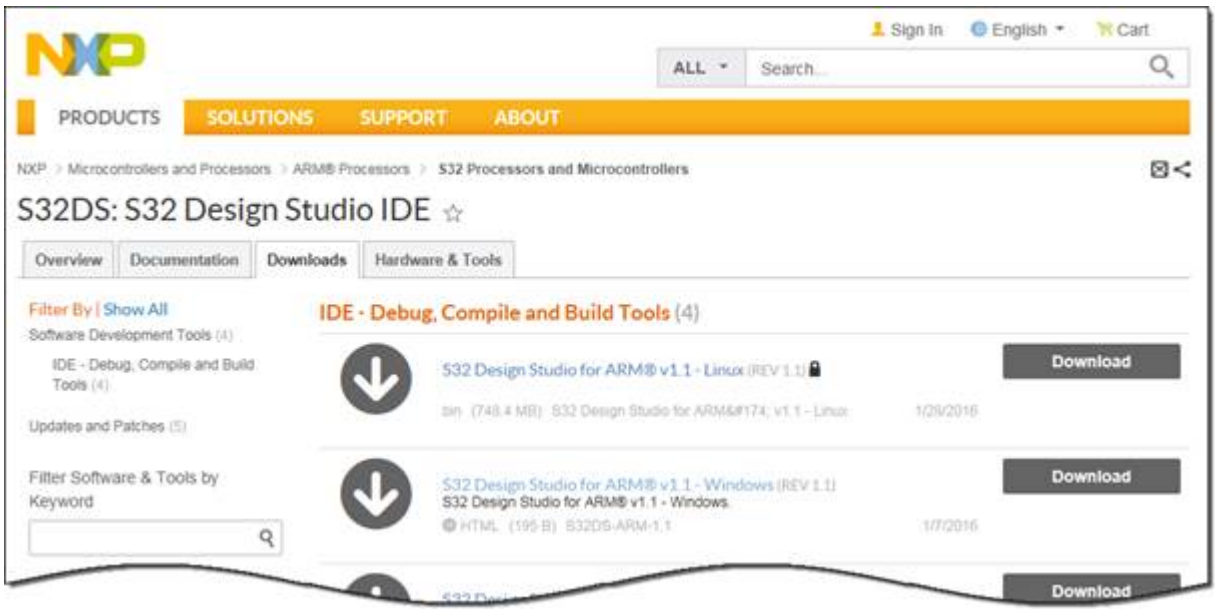
Installing S32DS Power v2017.R1

The S32DS Power v2017.R1 software development tools are installed on Windows using the Windows Installer. To install S32DS Power v2017.R1 using the Windows installer double-click the installer then the Windows Installer initiates. Follow the on-screen instructions and proceed through the installation.

You should not install S32DS Power v2017.R1 as administrator (with using of the **Run as administrator** option). The installer will raise user rights automatically when needed.

The S32DS Power v2017.R1 cannot be installed from the command line.

The updates for S32DS Power v2017.R1 are published on NXP web. Go to nxp.com/S32DS > Downloads tab > Updates and Patches. You will see it here:



For detailed description of installation process see **S32 Design Studio for Power Architecture, Version 2017.R1. Installation manual**.

Uninstalling S32DS Power v2017.R1

To uninstall S32DS Power v2017.R1 using the Windows uninstaller perform these steps:

1. Close the S32DS Power v2017.R1 application.
2. Navigate to the uninstaller:
   Windows **Start** > **All Programs** > **S32 Design Studio for Power Architecture, Version 2017.R1** > **Uninstall**.

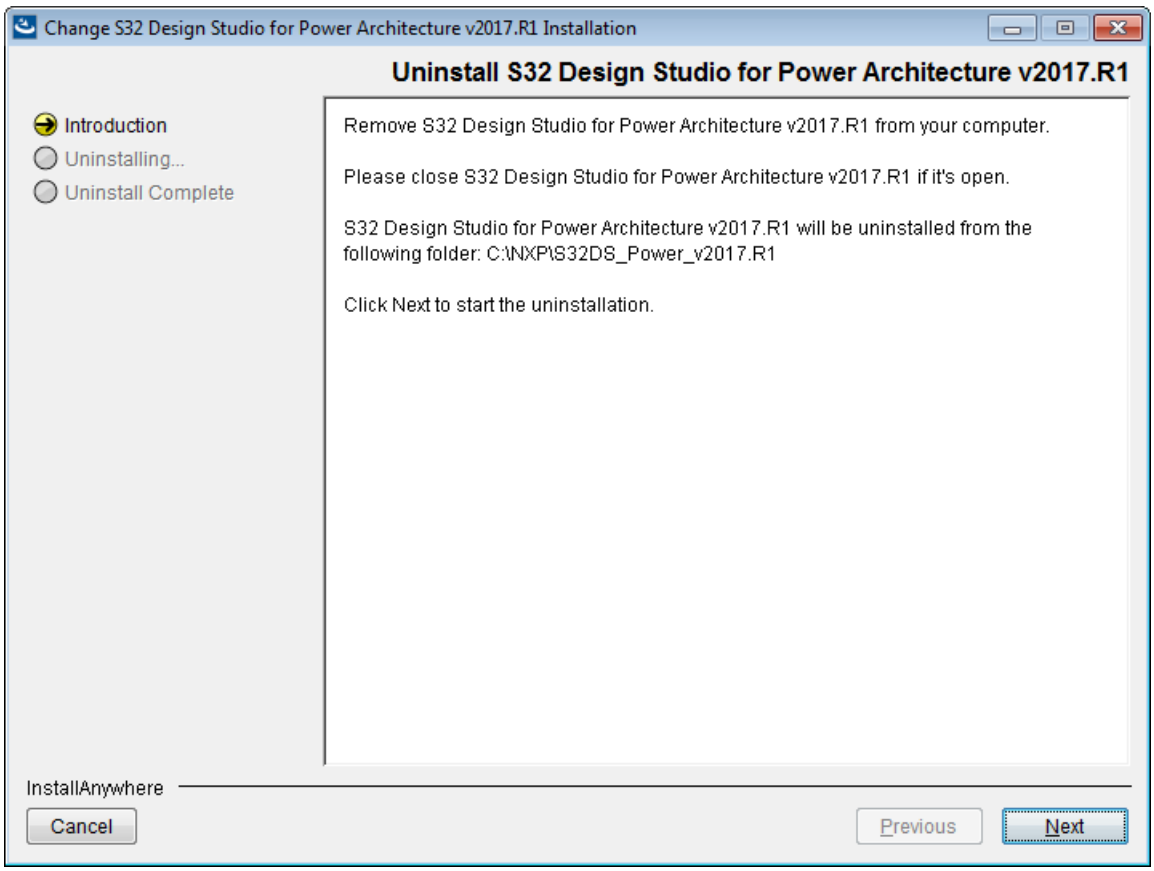
   Note:

   To uninstall S32DS Power v2017.R1 from Microsoft Windows 10:

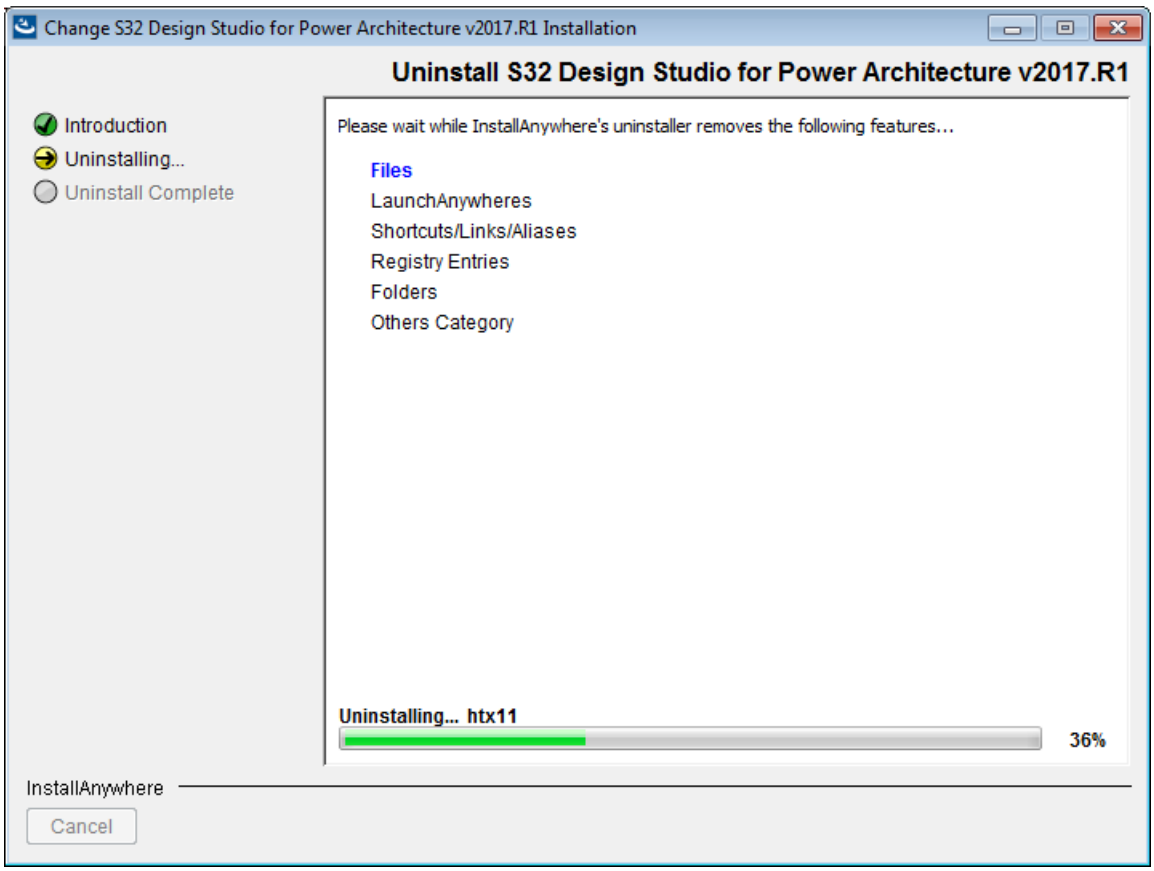
   1. navigate to the uninstaller: **Settings** > **System** > **Apps & features** > **S32 Design Studio for Power Architecture, Version 2017.R1**

   2. click the **Uninstall** button.

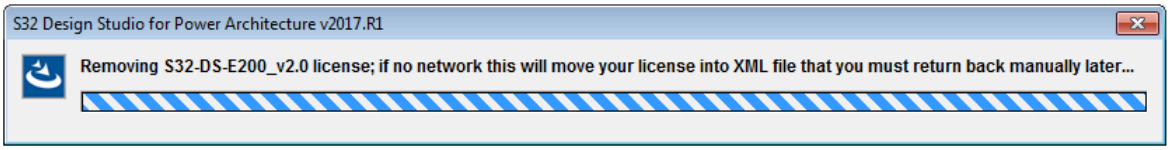
   The uninstaller initiates. Follow the on-screen instructions and proceed through the uninstallation.
3. When the uninstall wizard appears click the **Next** button:



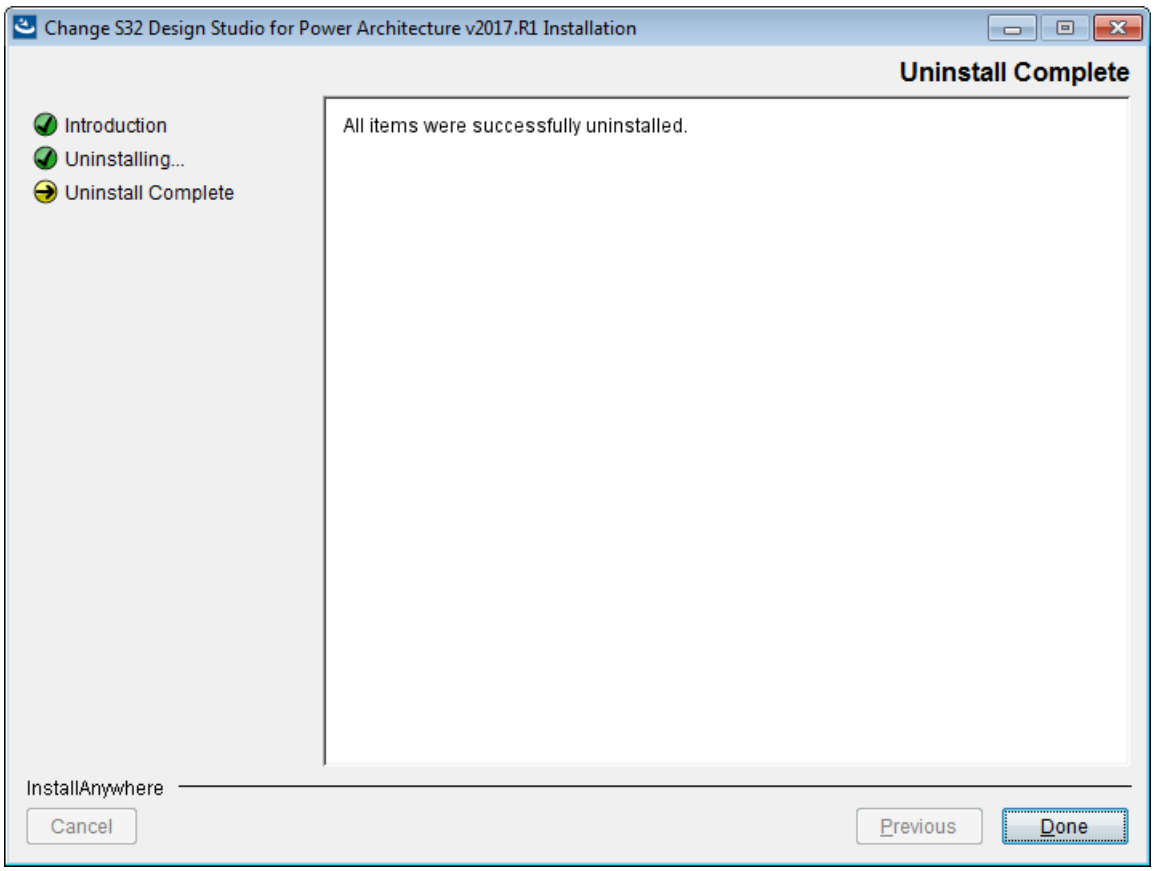
The uninstalling step opens. The S32DS Power v2017.R1 uninstallation starts:



Wait a few seconds while the S32DS Power v2017.R1 software uninstalls the license:



4. Wizard displays the **Uninstall complete** page. Click the **Done** button to close the wizard:



You don’t need to restart your system.

The uninstaller removes S32 Design Studio program group from the **Windows Start** > **All Programs** menu.

If there are some issues during the S32DS Power v2017.R1 removing then the wizard displays the corresponding messages on the **Uninstall complete** page.

Chapter 2

Working with Projects

Topics:

This chapter explains how to use the S32DS Power v2017.R1 to create and work with projects.

Types of Projects

This section describes types of projects you can create:

- S32DS Project (Elf S32DS project or Library project)
- Graph Tools Project.

S32DS Power v2017.R1 projects organize files and various compiler, linker, and debugger settings associated with the applications or libraries you develop. With S32DS Power v2017.R1, you can create a variety of projects that create ELF executable binary files that run directly on a given target board, without a Linux operating system. The type of project you create is based on selections you make in a wizard.

Note: With S32DS Power v2017.R1, you can't create projects that generate Linux ELF executable binary files for applications.

User can use the Eclipse Import wizard to import example projects into workspace. See details about importing files and projects in Common Features Guide (Import example project topic).

At the same time you can create the following types of S32DS or Graph Tools Projects:

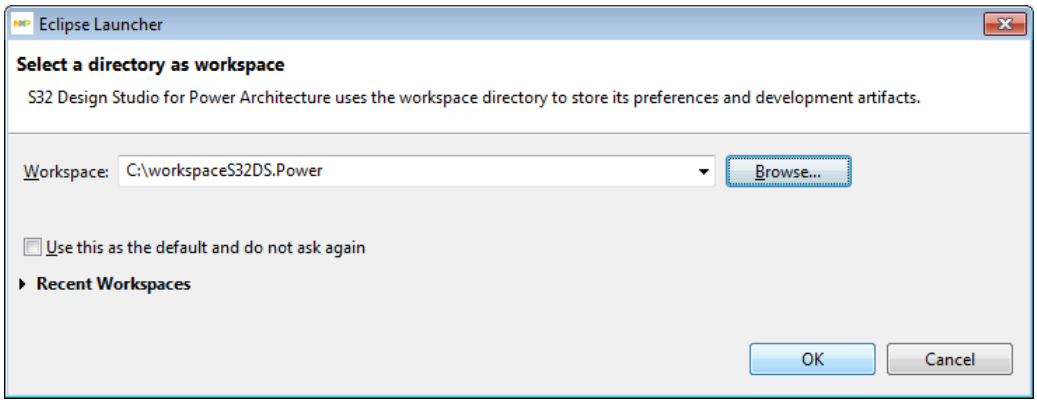
- single core projects
- multi-core projects.

Launching Workbench

To start S32 Design Studio for Power Architecture, Version 2017.R1 and begin working with it:

1. Select **Start** > **All Programs** > **NXP S32 Design Studio** > **S32 Design Studio for Power Architecture, Version 2017.R1** > **S32 Design Studio for Power Architecture, Version 2017.R1**. **Eclipse Launcher** dialog box appears and prompts you to select a workspace.

   Note: Workspace is a place on the file system where S32DS Power v2017.R1 stores projects that you create or import into the IDE.



2. Click **OK** to accept the default workspace and store your projects in the default location. To use a workspace different from the default, click **Browse**. **Select Workspace Directory** dialog box appears.
3. Specify the desired workspace. Select required directory or click **Make New Folder** to create a new directory for storing your projects. Click **OK**.

   Note:

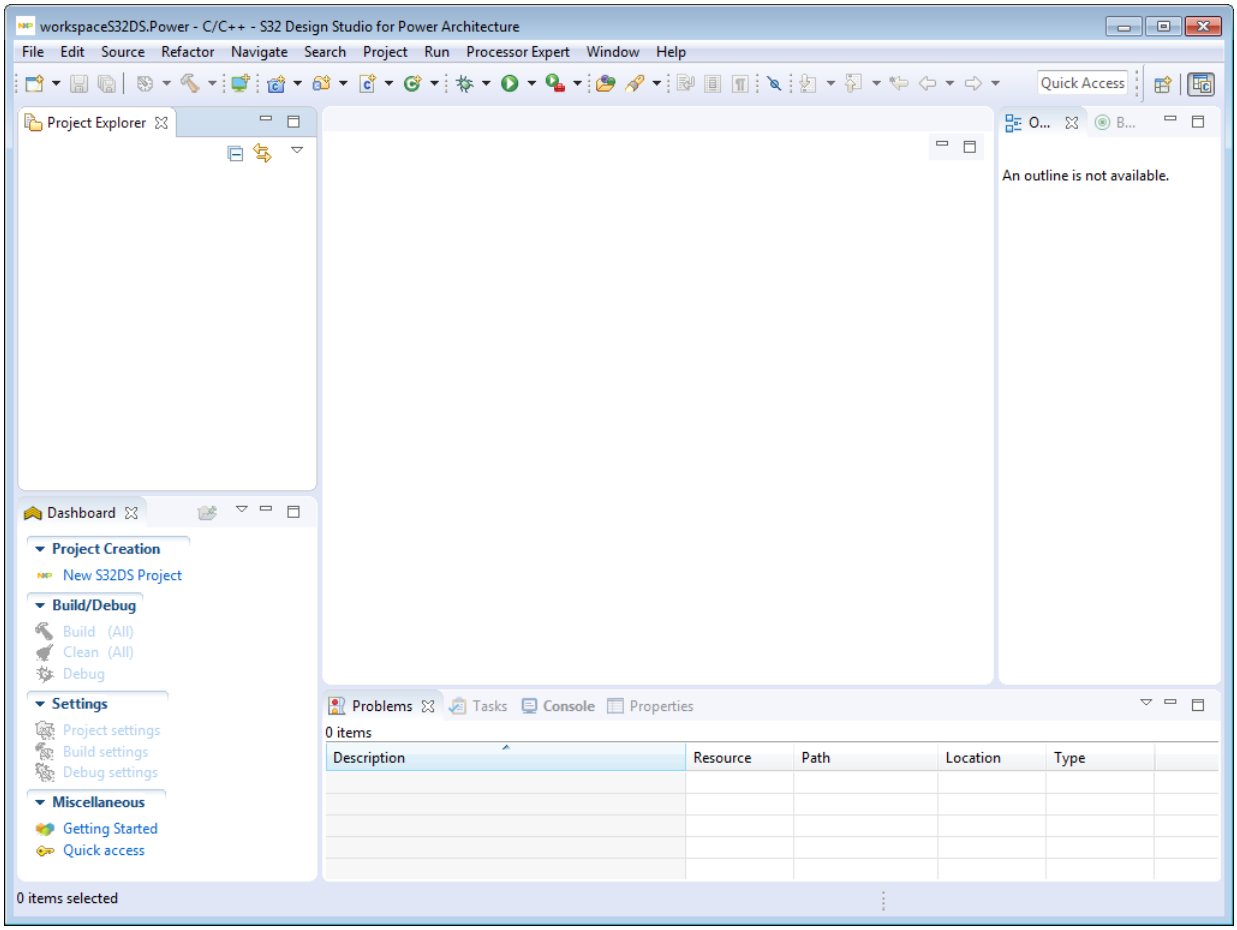
   Select the **Use this as the default and do not ask again** check box in the **Workspace Launcher** dialog box to set the chosen path as the default location for storing your projects.

   To change the default location of your workspace:

   1. Choose **Window** > **Preferences** to open the **Preferences** dialog window.

2. Find the **Workspaces** pane, and then select the **Prompt for workspace on startup** check box.
3. Click **Apply** to save changes.
4. Click **OK** to close the **Preferences** dialog window.

4. Click **OK**. S32DS Power v2017.R1 starts and the main IDE window appears:



New projects wizards

You can use **New S32DS Project** wizard or **New Graph Tools Project** wizard to create new single core or multi-core projects that group these files and settings into build and launch configurations.

A new project wizard helps you to quickly create new project. A new project wizard presents a series of pages that prompt you for the features and settings to be used when making your program. For example, the multi-coreproject options lets you select the processor you would like to use. This wizard also helps you specify the characteristics of the connection that communicates with a hardware target and other settings.

New S32DS Project wizard

This topic describes the various pages that the **New S32DS Project** wizard displays as it assists you in creating a project (**Elf S32DS** project or **Library** project).

The **New S32DS Project** wizard creates a project with the necessary startup code and linker configuration to bring the generated executable out of reset and into main with the supported run control solution.

The pages of the **New S32DS Project** wizard are:

1. Project name and type (the **New S32DS Projec**t page)

2. Cores and parameters for them (the **New S32DS Project for <processor name>** page)

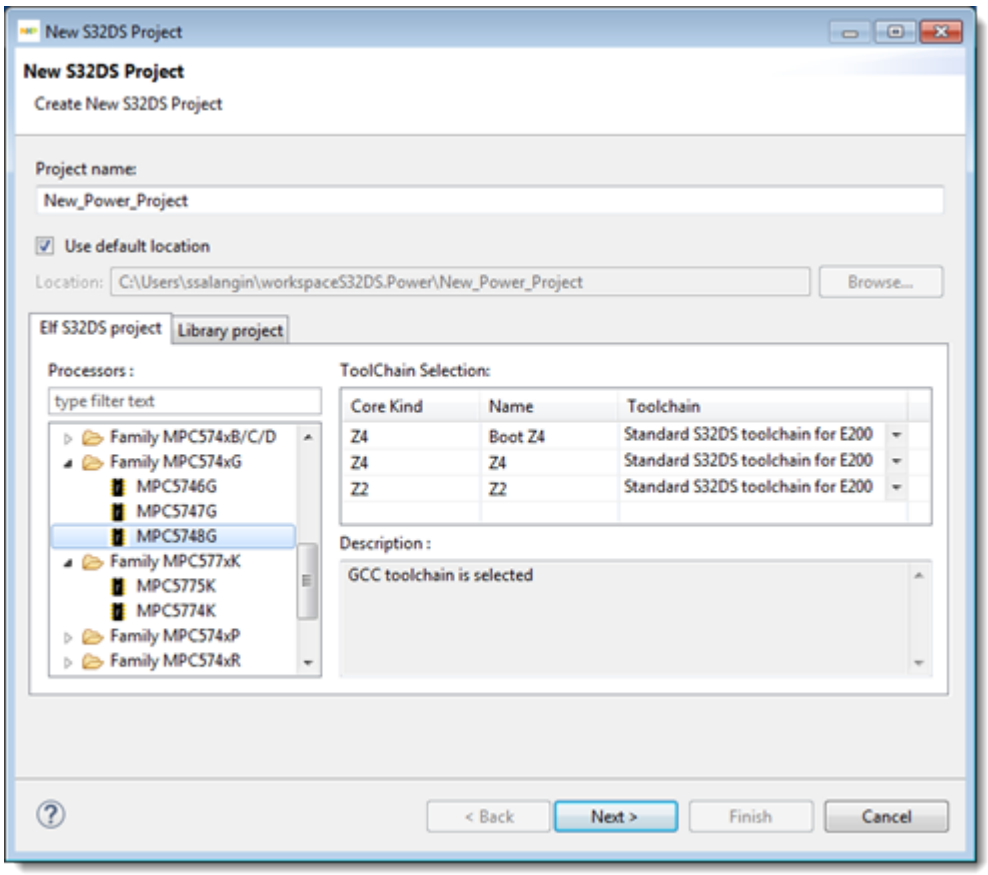
The pages of the **New S32DS Project** wizard can differ based on the project type or processor.

Project name and type

Use this page to specify the project name, the directory where the project files are located and processor. The table below describes the purpose of the various options.

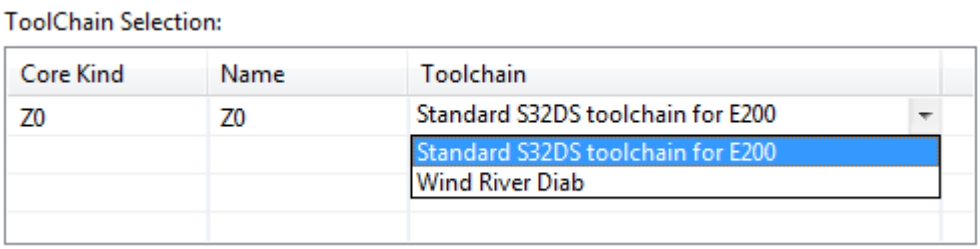
Table 3: New S32DS Project wizard – Page 1 settings

Option	Description
Project name	Enter the name for the new project in this text box. Do not use these reserved/special characters/symbols in the project name: < (less than) > (greater than) : (colon) / (forward slash) \ (backslash) | (vertical bar or pipe) ? (question mark) * (asterisk) " (double quote) @ (at) # (number sign) = (equals sign) ; (semicolon) " " (space) ' (single quote) ` (grave accent) {} (curly braces) () (parentheses) [] (square brackets) $ (dollar sign)
Use default location	Stores the files required to build the program in the Workbench's current workspace directory. The project files are stored in the default location. Clear the **Use default location** checkbox and click **Browse…** to select a new location.
Location	Specifies the folder that contains the project files. Click **Browse…** to navigate to the desired folder. This option is available only when **Use default location** checkbox is clear. Do not use the “Space” symbol in the folder name.
Elf S32DS project tab	Select the family and derivative or board you would like to use
ToolChain Selection table	The quantity of rows is equal to the quantity of cores in the processor selected in the **Processors** tree control. Select from the **ToolChain** drop-down list the compiler for new S32DS project: • **Standard S32DS toolchain for E200** (available for all processors) • **GHS PowerPC Standalone Executable Toolchain** (if the GHS plug-in was installed) • **Wind River Diab** (if the Diab plug-in was installed)
Library project tab	Select the family and derivative or board you would like to setup a project as library



After you select a processor and toolchain, next page of the wizard appears. Its options depend on the selected processor.

To select a toolchain use the **Toolchain** drop-down list:



The **GHS PowerPC Standalone Executable Toolchain** (GHS) and **Wind River Diab** (Diab) toolchains are available if corresponding Eclipse plug-ins were installed. See Toolchains from different vendors topic.

Cores and parameters for them

Use this page to select processors cores and parameters for them to build the project: the programming language etc.

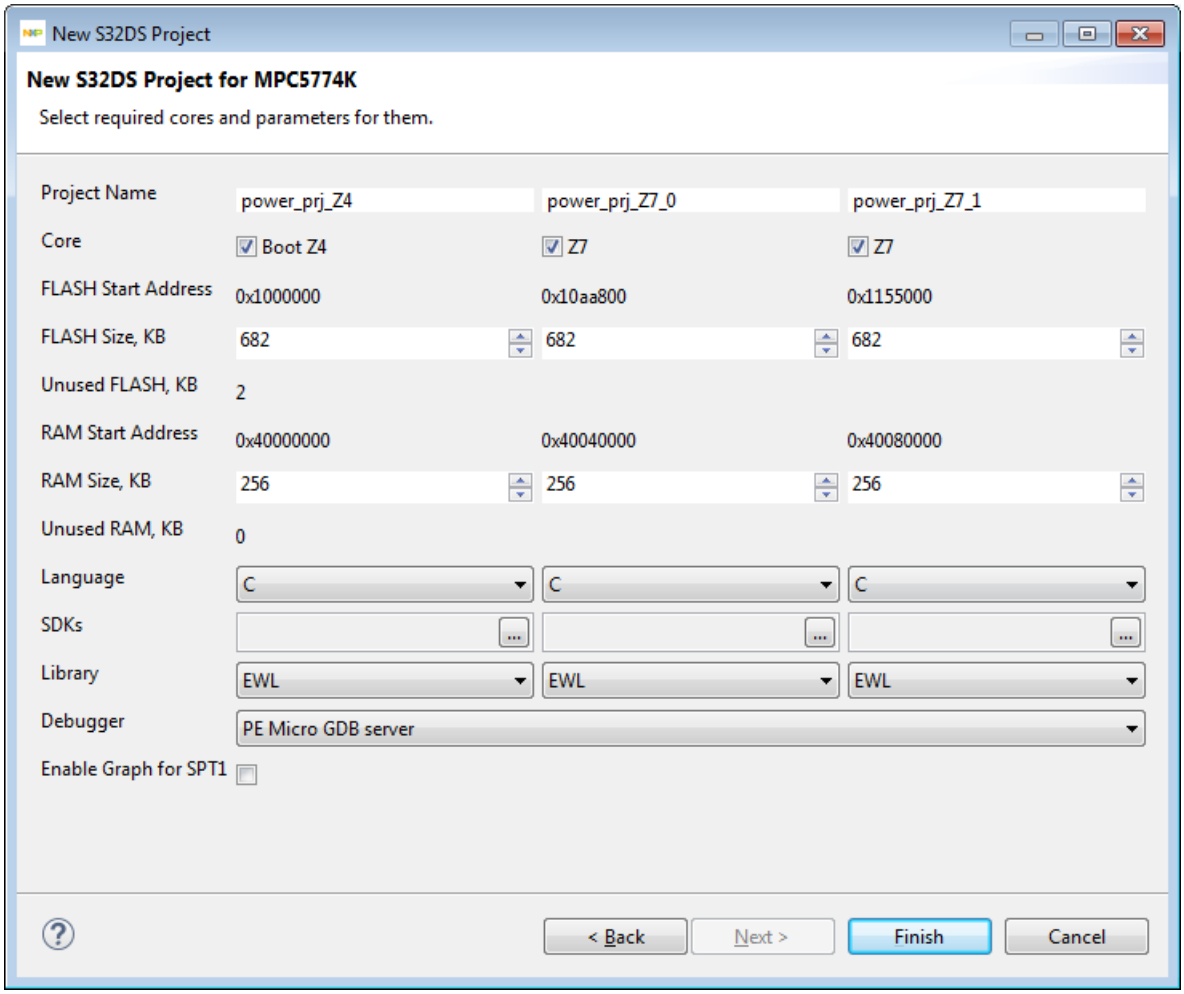
Table 4: New S32DS Project wizard – Page 2 settings

Option	Description
Project Name	See the project name for the core displayed in the **Project Explorer** pane. Read only.

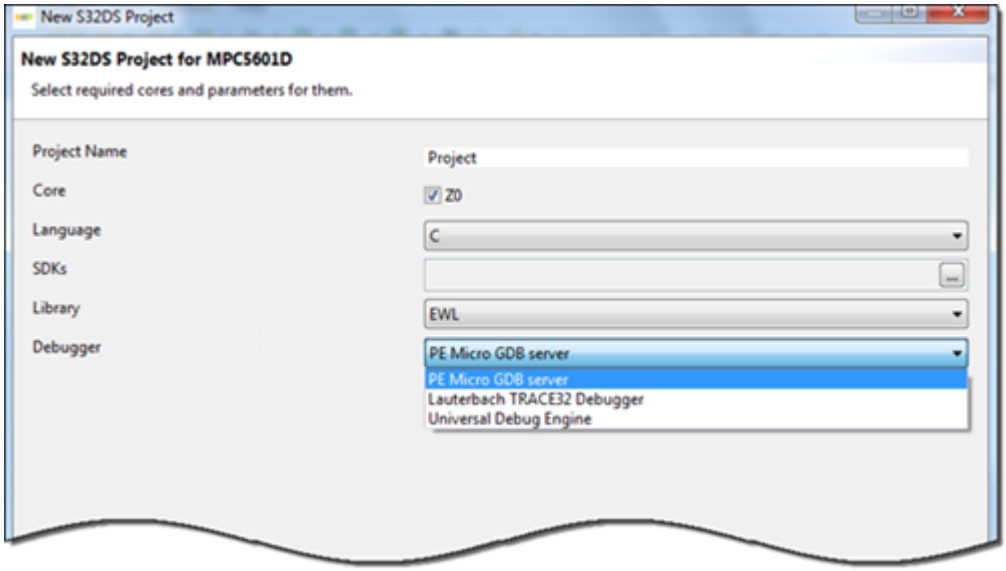
Option	Description
Core	Select required cores for project. The set of cores depends on the selected processor. By default, all core check boxes are selected.
FLASH Start Address	See the start address values of flash-memory for each core. Read only. The values depend on the selected FLASH size. Default values depend on the selected processor.
FLASH Size, KB	Use to specify size of flash-memory for each core (KB) with step 32. Default values depend on the selected processor.
Unused FLASH, KB	See the size of unused flash-memory for all cores. Read only. The value depends on the selected FLASH size. Default value depends on the selected processor.
RAM Start Address	See the start address of RAM -memory for each core. Read only. The values depend on the selected RAM size.
RAM Size, KB	Use to specify size of RAM-memory for each core (KB) with step 32. Default values depend on the selected processor.
Unused RAM, KB	See the size of unused RAM-memory for all cores. Read only. The value depends on the selected RAM size. Default value depends on the selected processor.
Language	The option you select sets up default compiler/linker options for the toolchain. Select the programming language that you want to use for writing the program's source code. You can select only one language: • **C** - sets up your application with ANSI C-compliant startup code, and initializes global variables • **C++** - sets up your application with ANSI C++ startup code, and performs global class object initialization. Default: **C**
SDKs	Use to select SDK. For more information, see chapter Working with SDKs. By default, this field is cleared.
Library	Use to specify support of the library linked to project: • **EWL** - c99 compliant Embedded Warrior Library • **EWL Nano** - a lightweight version of Embedded Warrior Library • **NewLib** - standard C/C++ library that is included with our e200 toolchain • **NewLib Nano** - a lightweight version of the standard C/C++ library. Default: **EWL** **Note:** The toolchain supports the **NewLib**, **NewLib Nano** and **Freescale EWL2** libraries but some of them are not accessible in the wizard.

Option	Description
I/O Support (only for single-core projects)	Single-core projects can support semi-hosting (debugger console). Select semi-hosting support to use for the project: • **No I/O** • **Debugger Console** Default: **No I/O**
Debugger	Select a connection type to use for the project: • **PE Micro GDB server** • **iSystem Debugger** (The iSystem Debug Plug-in for Eclipse should be installed to debug a project) • **Lauterbach TRACE32 Debugger** (the Lauterbach TRACE32 Eclipse plug-in should be installed to debug a project) • **Universal Debug Engine** (The UDE Eclipse plug-in should be installed to debug a project) Any of these debuggers can be used for any family. Default: **PE Micro GDB Server**
Enable Graph for SPT1	Select to create SPT1 graph tools project. **Note:** This option is available only for the MPC577xK processor family. By default, this check box is cleared.
Enable Graph for SPT2	Check to create SPT2 graph tools project. **Note:** This option is available only for the S32R274 processor family. By default, this check box is cleared.
Enable Graph for SPT2.5	Check to create SPT2.5 graph tools project. **Note:** This option is available only for the S32R372 processor family. By default, this check box is cleared.

Only one project is presented on the second step if a single core processor has been selected on the first step. A few projects can be presented on the second step if a multicore processor has been selected on the first step:



Use the **Debugger** drop-down list to select a connection type:



New SPT X Project wizard

This topic describes the page **New SPT X Project** wizard.[2] For detailed description of S32 Design Studio SPT Graphical Tool, see **S32 Design Studio for Power Architecture, Version 2017.R1 SPT Graphical Tool Reference Manual**.

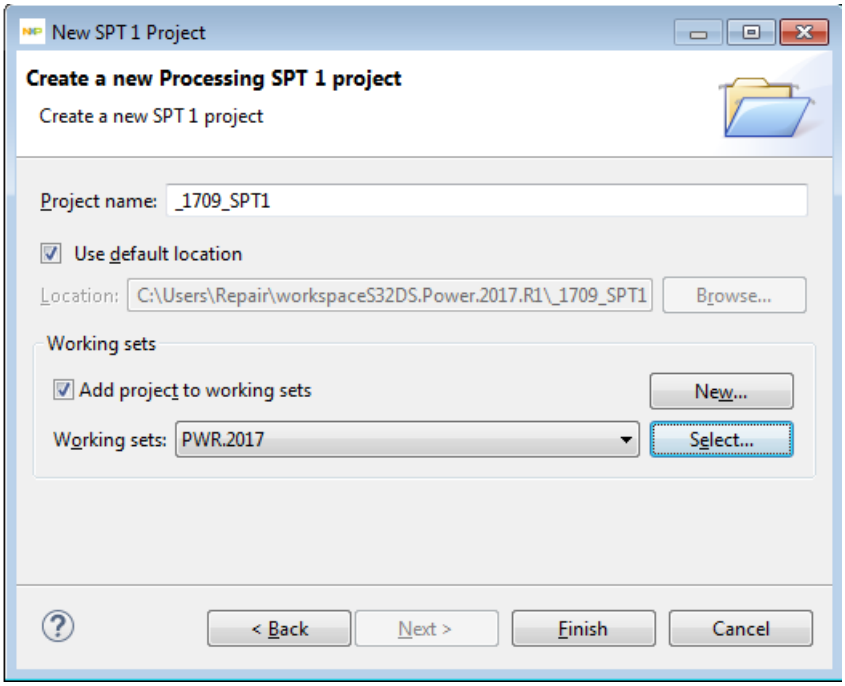
Use the **Create a new SPT project** page to specify the project name, the directory where the project files are located and representation. The table below describes the purpose of the various options.

Table 5: New SPT X wizard properties

Option	Description
Project name	A 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 program. If you create multiple projects within the same workspace, make sure to give unique names to projects that you include in this workspace. Enter the name for the new application project in this text box. Following identifiers are allowed: **Special characters**: _ (underscore) **Alphabetic characters**: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z **Numeric characters**: 0 1 2 3 4 5 6 7 8 9 Following special characters are not supported in project names: < (less than) > (greater than) : (colon) / (forward slash) \ (backslash) | (vertical bar or pipe) ? (question mark) * (asterisk) " (double quote) @ (at) # (number sign) = (equals sign) ; (semicolon) " " (space) ' (single quote) ` (grave accent) {} (curly braces) () (parentheses) [] (square brackets) $ (dollar sign) **Attention:** Do not start your names with a digit; you can use digits in postfixes.
Use default location	When selected, the IDE stores the project files in the current workspace directory, selected on the start of the IDE. Clear the **Use default location** check box, and then click **Browse…** to select a new location.
Location	Specifies location of the workspace in which you want the project files to be stored. Click **Browse…** to navigate to the desired directory. By default, the field contains the path to the current workspace. This filed is unavailable until you clear the **Use default location**.

[2] Here X represents a substitution for the type of an SPT project: 1, 2, or 2.5

Option	Description
Add project to working sets	When selected, the IDE adds the new project to the selected working sets.
Working sets	Specifies the working set where you want the created project to be placed. Click **Select…** to select desired *working sets*, groups of projects within the same workspace. The list is unavailable until you select the **Add project to working sets** check box.



Creating projects

The following topics explain the steps to create S32DS Power v2017.R1 projects.

A wizard generates a set of projects with placeholder files and specific settings (build and launch configurations) for selected target. After the projects have been created, you can easily change any generated setting to suit your needs.

Creating S32DS Projects

To create a new S32DS project for a processor using the **New S32DS Project** wizard:

1. Launch the Workbench.
2. Select **File** > **New** > **New S32DS Project** from the IDE menu bar. The first page of the **New S32DS Project** wizard appears.
3. Specify a name for the new project in the **Project name** text box.
4. Expand the **Processors** tree control and select the processor you would like to use. For example, **Family MPC574xG** > **MPC5748G**.
5. Expand the **Toolchain** list and select the toolchain you would like to use.

   Note: Availability of the options depends on the processor you selected.
6. Click **Next**. The second page of the wizard appears.
7. Expand the controls and select following project settings:

Core	(use checkbox)	
FLASH Start Address	(read only)	
FLASH Size, KB	(enter or select)	
Unused FLASH, KB	(read only)	
RAM Start Address	(read only)	
RAM Size, KB	(enter or select)	
Unused RAM, KB	(read only)	
Language	(expand drop down list)	
SDKs	(use the Select SDK window)	
Library	(expand drop down list)	
I/O Support	(expand drop down list)	(only for single-core projects)
Debugger	(expand drop down list)	
Enable SPT1	(use checkbox)	(only for MPC577xK microcontrollers)
Enable SPT2	(use checkbox)	(only for S32R274 microcontroller)
Enable SPT2.5	(use checkbox)	(only for S32R372 microcontroller)

Note: Default values of the options depend on the processor you selected.

8. Click **Finish**. The wizard creates the new project according to your specifications. You can access the project from the **Project Explorer** view in the Workbench window. The new project is ready for use. You can now customize it by adding your own source code files, changing debugger settings, or adding libraries.

   Note: You can click **Cancel** at any step in the wizard to stop the project creation, discard all changes and close the wizard dialog box.

Creating SPT X Project

The following section demonstrates how you can create a new SPT X[3] project.

To create a new SPT X project, follow the procedure.

1. In the main S32DS Power v2017.R1 window, do one of the following:.

Option	Action
Use the menu	Select **File** > **New** > **Other…** in the menu bar
Use the keyboard	Press **Ctrl**+**N**

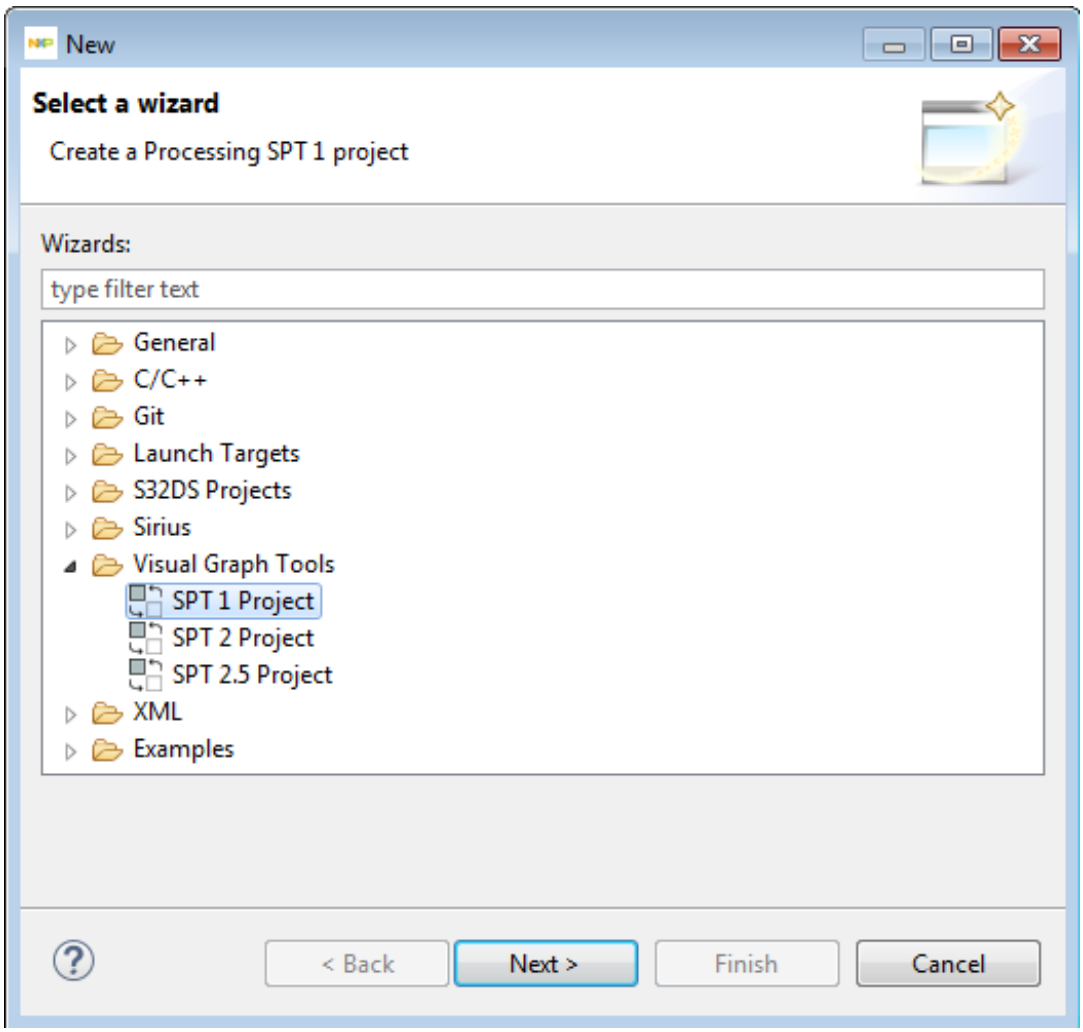
   The project creation wizard starts.

2. On the **Select a wizard** page, do one of the following:

Option	Action
Use the filter	In the **Wizards** field, type SPT
Use the tree	Navigate to the **Visual Graph Tools** folder, and then expand it

3. Click one of the items to create a specific SPT project, and then click **Next**.
   The **Create a new Processing SPT** project page opens.

[3] Here X represents a substitution for the type of an SPT project: 1, 2, or 2.5

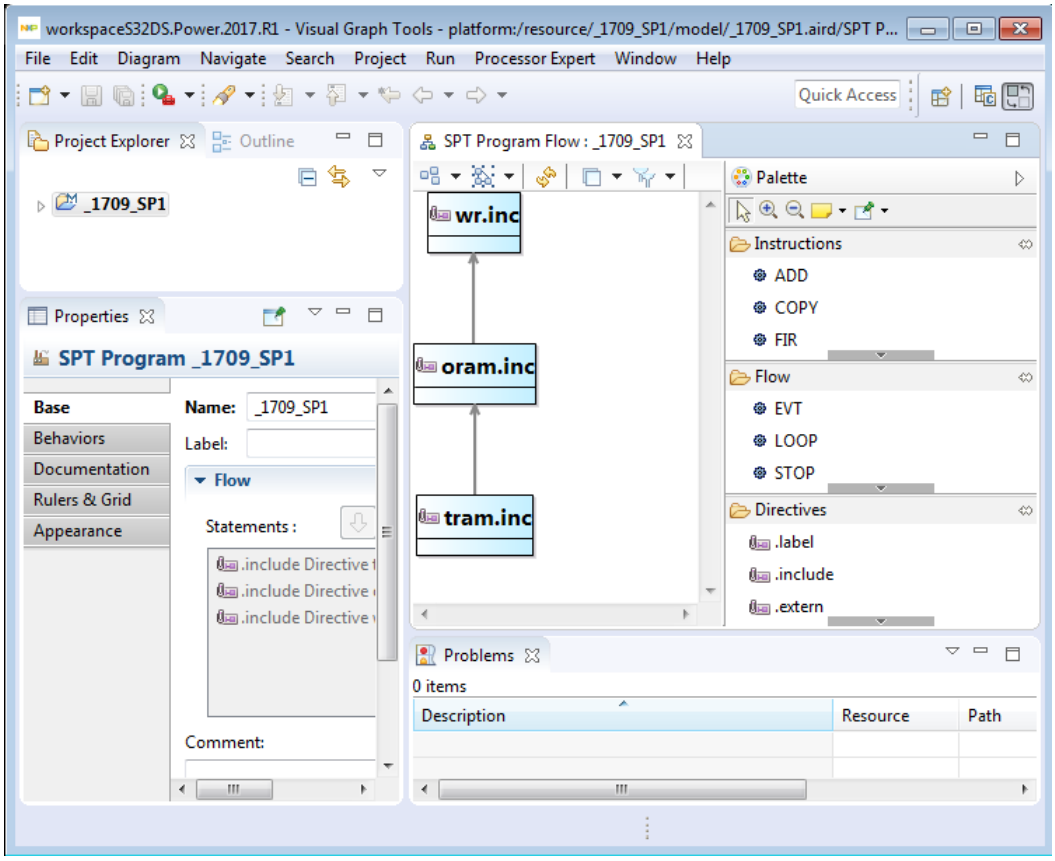


4. Give the project a unique name and specify where you want the project to be stored.
   a) In the **Project name** field, specify the name for the project you are creating.

      Important: The name must be a valid C identifier and must be unique.
   b) Clear the **Use default location check box**, and then click the **Browse** button to specify a custom location of the workspace where you want to store this project.

      Note: By default, created projects are saved to the workspace selected upon the start of the IDE.
   c) Select the **Add project to working sets**, and then select a working set in the list below, if you want to place the project in a specific *working set*, a group of projects within a workspace.
5. Click **Finish** to complete the wizard.
   a) Optional: In the **Open Associated Perspective** message box, click **Yes** to switch to the window perspective with the tools specific to Visual Graph Tools after S32DS Power v2017.R1 creates the project.

Created project will show in the Project Explorer, and a new tab will open with the automatically generated default diagram.

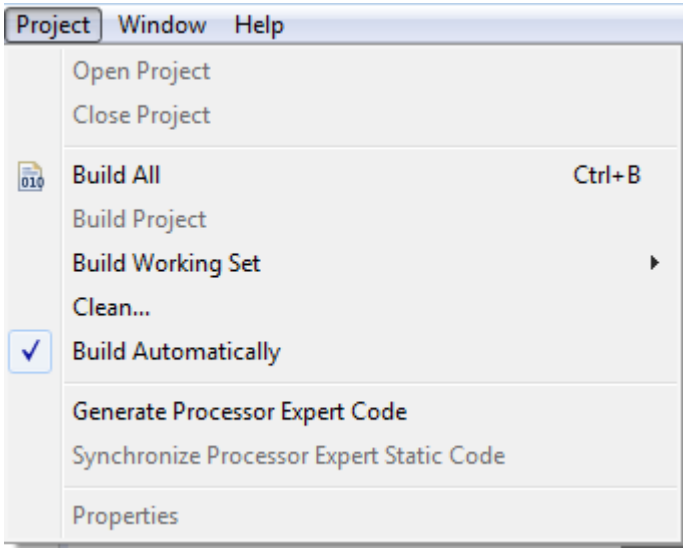


Building projects

The recently built S32DS Power v2017.R1 project is pre-configured and you can easily build the project for your target board. However, if you want to change the configuration of the project you can adjust the build properties. For more information on build properties, see Build Properties for S32DS Projects.

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

To build only the selected projects, and any prerequisite projects that need to be built in order to correctly build the selected projects, select **Project** > **Build Project** from the S32DS Power v2017.R1 menu bar or right-click on a selected project and select **Build Project** or click .



Alternatively (to build all projects), select **Project** > **Build All**.

Monitor the generated command lines used to build the embedded application in the build **Console** view. Any problems with the build will be reported under the **Problems** view. Assuming the build is successful, the generated binary will be listed under the project in the **Project Explorer** view.

Debugging projects

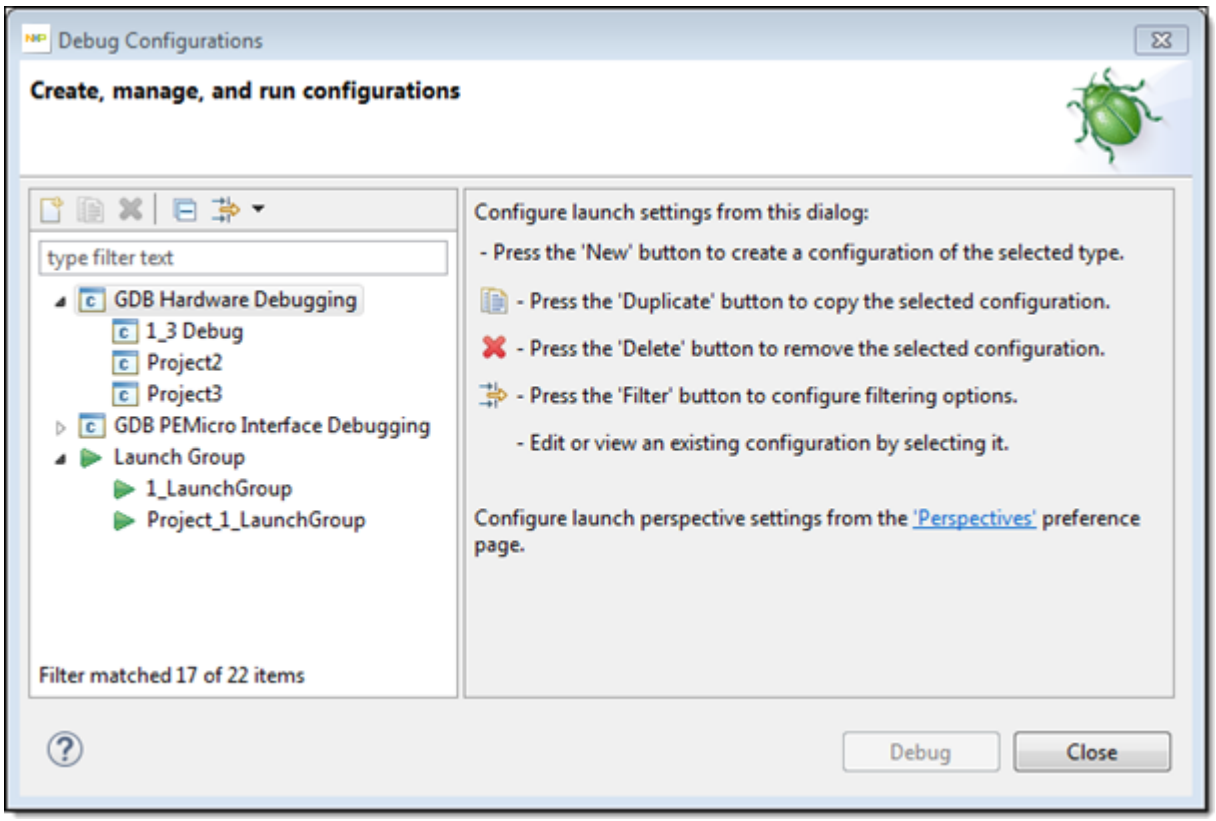
When you use the **New S32DS Project** wizard to create a set of projects, the wizard sets the debugger settings of the project's launch configurations to specific values. You can change these generated values based on your requirements.

To debug a project, perform these steps:

1. Open the **Debug** perspective.
2. From the main menu bar of the S32DS Power v2017.R1, select **Run** > **Debug Configurations…**. Alternatively, you can click > **Debug Configurations…**. The **Debug Configurations** dialog box appears. The left side of this dialog box has a list of debug configurations that apply to the current application.

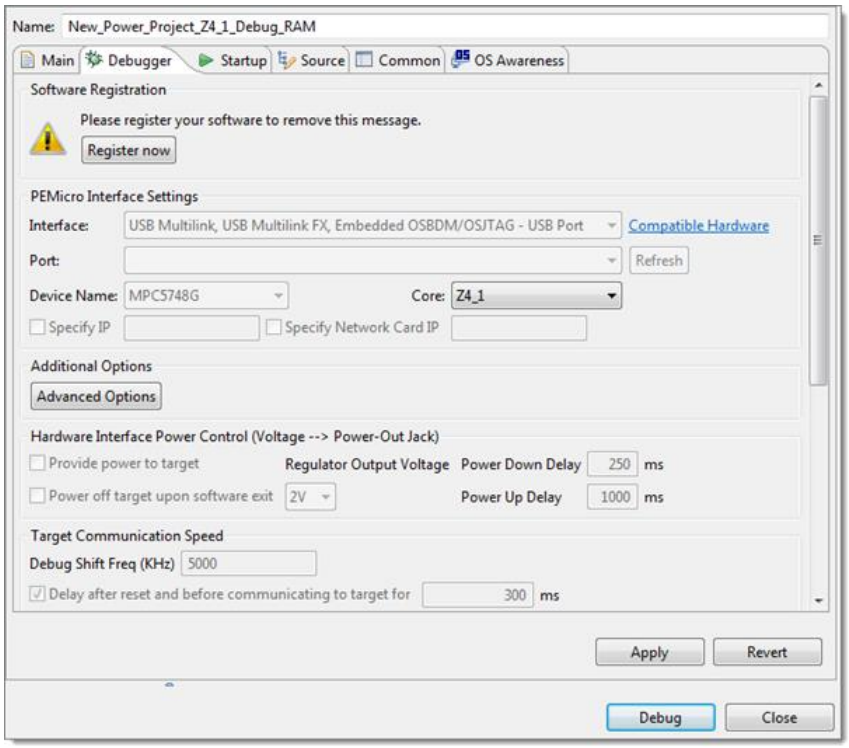
   Note: For more information on how to use the debugger, refer to the **S32DS Power v2017.R1 Common Features Guide** and the Working with Debugger chapter of this manual.
3. Expand the tree.
4. From the expanded list, select the debug configuration that you want to modify.

   The figure below displays the **Debug Configurations** dialog box:



5. In the **Main** tab, ensure that the correct **Project** and **C/C++ Application** are selected.
6. Click the **Debugger** tab.

   The **Debugger** tab opes showing debugger settings.



7. Based on the debug interface you have selected, change the appropriate debugger settings.

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

   Note: For more information on debugger, refer to the chapter Working with Debugger.

9. Click **Apply** to save the new settings.
10. Click **Debug** to start the debugging session. The S32DS Power v2017.R1 uses the settings in the launch configuration to generate debugging information and initiate communications with the target board.

    Note: If the **Debug** perspective is not open, you will be prompted to open the **Debug** perspective. Select **Yes** to switch perspective. The **Debug** perspective will open and the embedded application will break on the breakpoint set on main.

    You just finished starting a debugging session in the **Debug** perspective and attaching the debugger to a process.

    Note:

    If default USB-port ID in debug configuration is not corresponds to USB-port ID of the connected device then S32DS Power v2017.R1 cannot debug a created project successfully. For example the "No source available" alert message can appears in the IDE.

    To fix the problem please follow the instruction below:

    1. Open the **Debug Configuration** window.

    2. Select used launch configuration.

    3. Open the **Debugger** tab.

    4. Check the **Port** value, then check whether you have your device connected to PC via USB or Serial or Ethernet. If the port has assigned value and the **Apply** button is enabled, it means that your device was recognized and the setting was updated.

    5. Click **Apply** then **Debug** buttons. You should get debugging started correctly.

    For detailed descriptions of debug configuration for GDB *PEMicro* interface please refer to **P&E GDB Server Plug-In for Kinetis Devices. Debug Configuration User Guide** in the S32DS Power v2017.R1 Help (S32DS Power v2017.R1 > Common Manuals).

    You can click **Revert** to undo any of the unsaved changes. The S32DS Power v2017.R1 restores the last set of saved settings to all pages of the **Debug Configurations** dialog box. Also, the IDE disables **Revert** until you make new pending changes.

Closing and deleting projects

Close unused projects. Eclipse caches files for all open projects in the workspace. If you need multiple projects open, try to limit the number of projects to no more than 10.

To delete a project, follow these steps.

1. Select the project you want to delete in the S32DS Power v2017.R1 **Project Explore**r view.
2. Select **Project** > **Delete**.

   The **Delete Resources** dialog box appears.

   Note: Alternatively, you can also select **Delete** from the context menu when you right-click on the project.
3. Check the **Delete project contents on disk (cannot be undone)** checkbox if you want to delete the contents of the selected project. Else, clear the **Delete project contents on disk (cannot be undone)** checkbox.

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

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

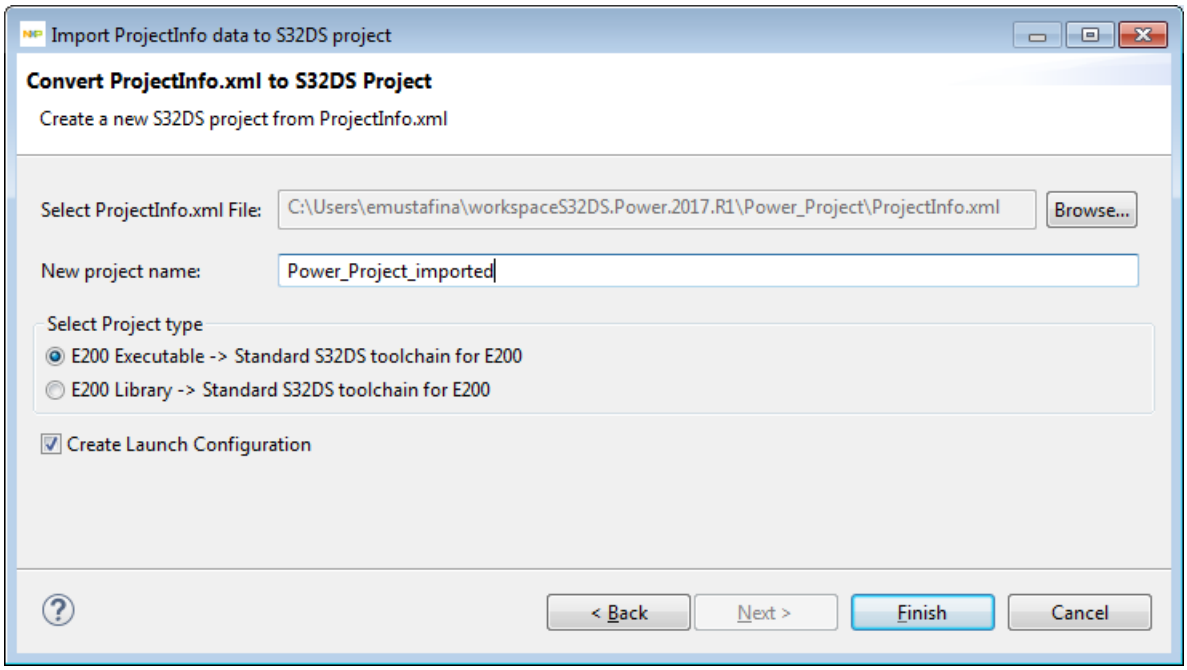
Note: To clean the project, right click on the project in the **Project Explorer** view and select **Clean Project**. Once cleaned select **Build Project.**

Import of Project info file

This section explains how to import an existing **ProjectInfo.xml** file to S32DS Power v2017.R1. User can import into the S32DS Power v2017.R1 by using the **Import** wizard. The **Import** wizard lets you import example project into the workspace.

To import a **ProjectInfo.xml** file:

1. Select **File** > **Import…**from the menu bar. The Import wizard appears.
2. Expand the tree control and select **S32 Design Studio** > **ProjectInfo.xml Importer**. Click **Next**. The **Import Project Info data to S32DS project** window appears with file import options.



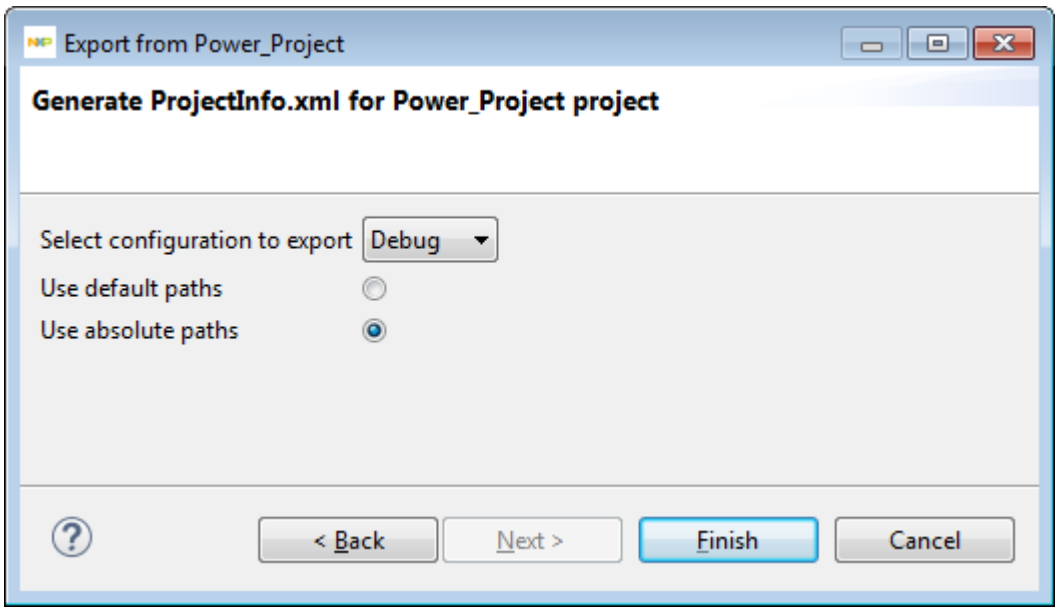
3. Select the Project Info file which you would like to import, enter new project name.
4. Select the Project type (executable or library).
5. Click **Finish**. New project will be created in your workspace.

Export of Project Info

This section explains how to export Project Info to **ProjectInfo.xml** file from S32DS Power v2017.R1. User can export to the file by using the **Export** wizard. The **Export** wizard lets you create Project Info file into the workspace folder.

To export Project Info:

1. Select a project or some projects for export to **ProjectInfo.xml** files.
2. Select **File** > **Export…** from the menu bar. The **Export** wizard appears.
3. Expand the tree control and select **S32 Design Studio** > **Project Info Export Wizard**. Click **Next**. The Generate **ProjectInfo.xml** page appears:



4. Select the configuration to export and paths parameters to export Project Info.
5. Click **Finish**. Selected Project Info will be exported to the **ProjectInfo.xml** files in your workspace.

Chapter 3

Build properties for projects

Topics:

This chapter explains build properties for a project. The **New S32DS Project** wizard uses the information it gathers from you to set up the project's build settings and debug configurations.

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

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

When the **New S32DS Project** wizard completes its process, it generates debug configurations with names that follow the pattern:

<projectname>_<configtype>_<targettype>, where:

- **<projectname>** - represents the name of the project
- **<configtype>** - represents the build configuration's name
- **<targettype>** - represents the type of target hardware on which the debug configuration acts

For each project, you can specify build settings, such as:

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

Changing Build Properties

The **New S32DS Project** wizard creates a set of build properties for the project. You can modify these build properties to better suit your needs.

Perform these steps to change build properties:

1. Start the S32DS Power v2017.R1.
2. In the S32DS Power v2017.R1 **Project Explorer** view, select the project for which you want to modify the build properties.
3. Select **Project** > **Properties** from the S32DS Power v2017.R1 menu bar. The **Properties for <project name>** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.
4. Expand the **C/C++ Build** property. The **C/C++ Build** pane appears on the right.
5. Use the **Configuration** drop-down list (on the right pane) to specify the build configuration for which you want to modify the build settings:
6. Change the settings that appear in the pane.
7. Click the **Apply** button at the bottom of the right pane. The S32DS Power v2017.R1 saves your new settings. You can select other tool pages and modify their settings.
8. When you finish, click **OK** to save your changes and close the **Properties for <project name>** window.

Restoring build properties

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

This changes the values of the options to the absolute default of the toolchain. The **Restore Defaults** button changes the values to the toolchain-default NOT to the specific values were set after the project creation was finished.

For example, when a project is created the **Tool Settings** > **Target Processor** panel has some values set, which are specific to the project. Clicking the **Restore Defaults** button defaults the values of options.

Defining C/C++ Build Settings

The **Properties for <project>** > **C/C++ Build** page includes all builder-specific property pages.

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

To define C/C++ build settings, perform these steps:

1. Start the S32DS Power v2017.R1.
2. In the S32DS Power v2017.R1 **Project Explorer** view, select the project for which you want to modify the builder settings.
3. Select **Project** > **Properties**.

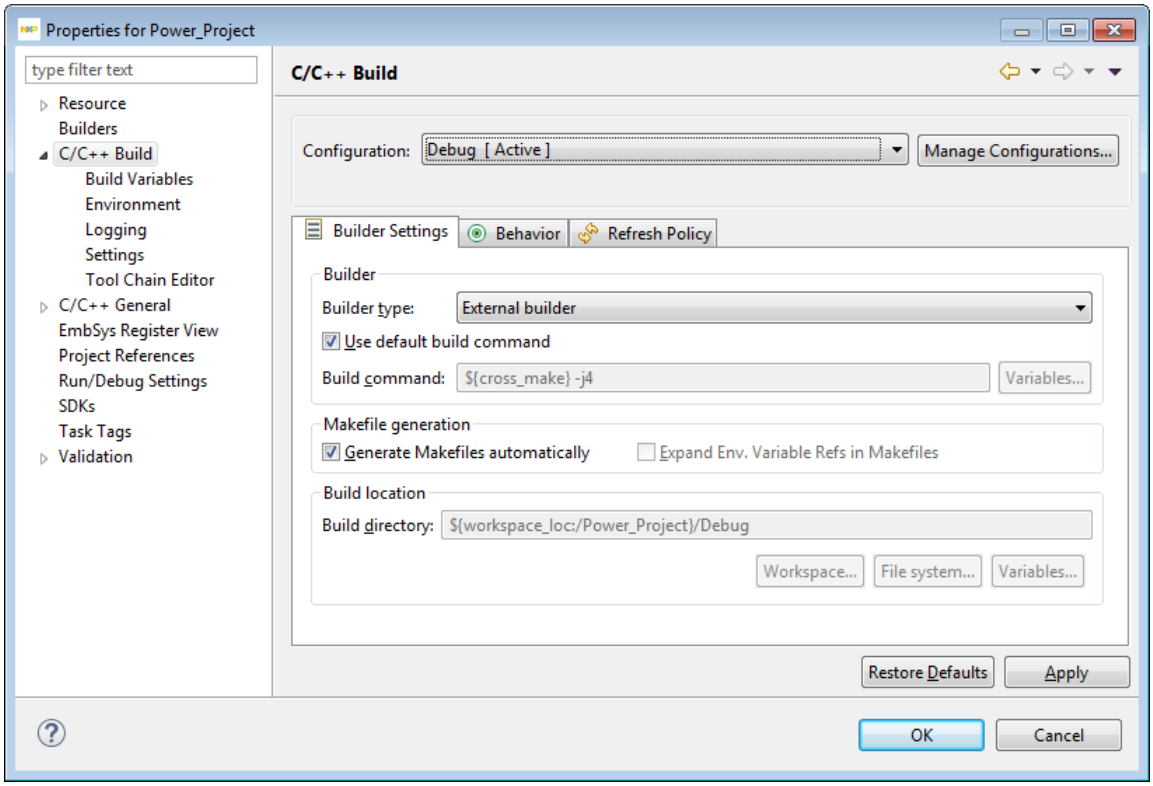
   The **Properties for <project name >** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.
4. Select **C/C++ Build**.

   The **C/C++ Build** page appears. Select one of the tabs:

   - **Builder Settings** to define builder settings
   - **Behavior** to define build behavior

Define Builder Settings

To define builder settings, on the **C/C++ Build** page click the **Builder Settings** tab.



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

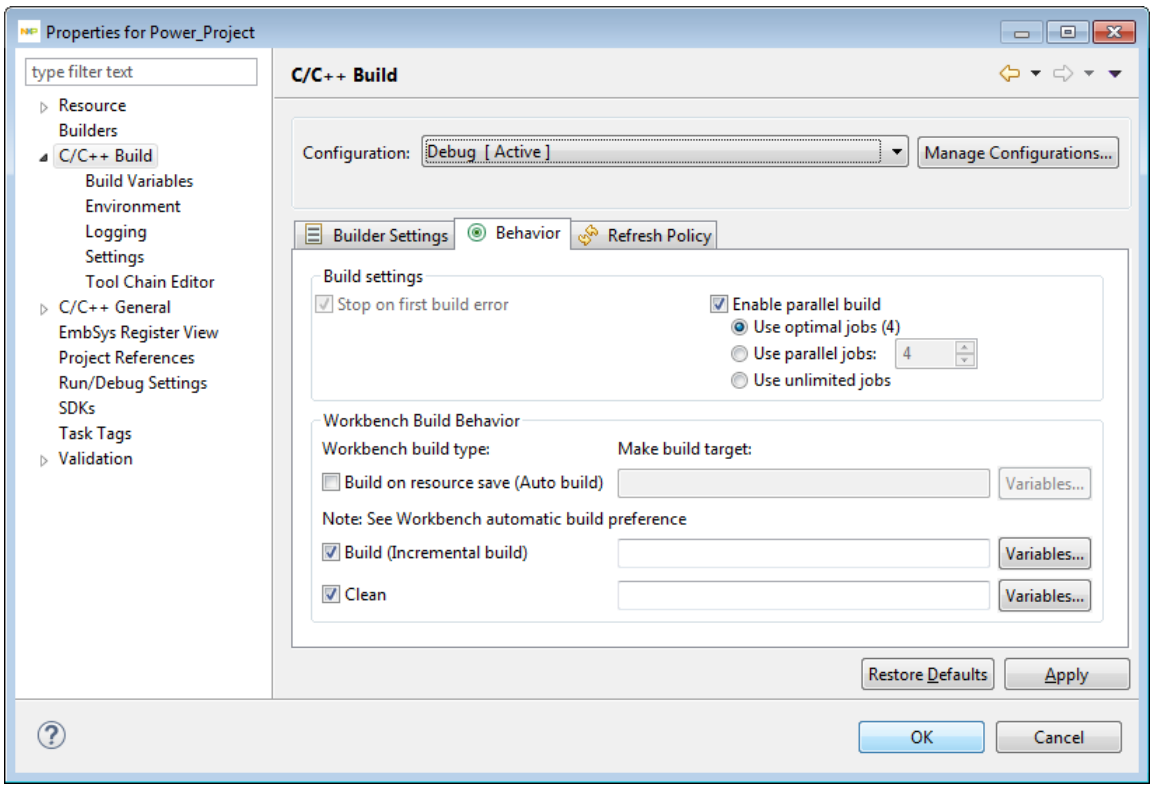
Table 6: Builder Settings Options

Group	Option	Description
	Configuration	Specifies the build configurations for the selected project.
	Manage configurations	Click to open the **Manage Configurations** dialog box that lets you set configurations based on the specified toolchains of the selected project. You can also create new build configurations, rename an existing configuration, or remove the ones that are no longer required.
Builder	Builder type	Specifies the type of builder to use: • **Internal builder** – Builds C/C++ programs using a compiler that implements the C/C++ Language Specifications. • **External builder** – External tools let you configure and run programs and Ant buildfiles using the Workbench, which can be saved and run at a later time to perform a build.
	Use default build command	Check to indicate that you want to use the default make command. Clear when you want to use a new make command. This option is only available when the **Builder type** option is set to **External**.

Group	Option	Description
	Build command	Specifies the default command used to start the build utility for your specific toolchain. Use this field if you want to use a build utility other than the default make command.
	Variables…	Click to open the **Select build variable** dialog box and add the desired environment variables and custom variables to the build command.
Makefile generation	Generate Makefiles automatically	Check to enable S32DS Power v2017.R1 change between two different CDT modes: it either uses the customer's makefile for the build, if one exists, or it generates makefiles for the user.
	Expand Env. Variable Refs in Makefiles	Check to define whether environment variables should be expanded in makefile.
Build location	Build directory	Specifies the location where the build operation takes place. This location will contain the generated artifacts from the build process. This option appears disabled when the **Generate Makefiles automatically** option is enabled.
	Workspace…	Click to open the **Folder Selection** dialog box and specify the build directory by selecting from *workspace*. The directory will contain all artifacts created during project build.
	File system…	Click to open the **Browse For Folder** dialog box and specify the build directory by selecting it from *file system*.
	Variables…	Click to open the **Select build variable** dialog box and select a variable to specify as an argument for the build directory, or create and configure simple build variables which you can reference in build configurations that support variables.

Define Build Behavior

To define build behavior, on the C/C++ Build page click the Behavior tab.



The build behavior settings for the selected build configuration appear. The table below describes the build behavior options.

Table 7: Behavior Options

Group	Option	Description
Build settings	Stop on first build error	Check to stop building when S32DS Power v2017.R1 encounters an error. Clearing this option is helpful for building large projects as it enables make to continue making other independent rules even when one rule fails.
	Enable parallel build	Check to activate the generation of parallel builds. However, you need to determine the number of parallel jobs to perform: • **Use optimal jobs number** - Lets the system determine the optimal number of parallel jobs to perform. • **Use parallel jobs** – Lets you specify the maximum number of parallel jobs to perform. • **Use unlimited jobs** – Lets the system perform unlimited jobs.
Workbench Build Behavior	Workbench build type	Specifies the builder settings when instructed to build, rebuild, and clean.

Group	Option	Description
	Build on resource save (Auto build)	Check to build your project whenever resources are saved. By default, this option is selected and builds occur automatically each time resources are modified. Clear if you do want that the build occurs only manually using a menu item.
	Build (Incremental build)	Defines what the standard builder will call when an incremental build is performed.
	Variables	Click to open the **Select build variable** dialog box and add variables to the make build target command.
	Clean	Defines what the standard builder calls when a clean is performed. The make clean is defined in the makefile.

C/C++ Build Tool settings

This release of gcc for e200-VLE supports the "Power Architecture 32-bit Application Binary Interface Supplement 1.0 - Embedded". It is based on gcc 4.9.2, binutils 2.24 and gdb 7.8.2.

The release supports VLE codegen for:

- e200z0
- e200z2
- e200z3
- e200z4
- e200z6
- e200z7

It has built-in MULTILIB support for the e200 with soft-float or the Embedded FPU, libraries also support single precision double folding (**-fshort-double**) and the SPE.

This distribution provides support for newlib, newlib-nano and Freescale EWL2. Since distinct libraries are provided, the tools require setting explicitly **-sysroot** to perform MULTILIB resolution:

- –sysroot=INSTALL/powerpc-eabivle/newlib for newlib
- –sysroot=INSTALL/ewl2 for EWL2

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

To define C/C++ build tool settings, perform these steps:

1. Follow the steps listed in Section Changing Build Properties.
2. Select from build properties list the Settings property. The Settings pane appears on the right. The Tool Settings tab is opened by default.
3. From the list of tools on the left of Tool Settings pane, select the tool for which you want to modify properties.

   Set of tool settings depends from language used for project: C or C++. Default values depend from debug configuration selected from the Configuration list.

Cross Settings

This section describes cross settings that can be configured in the build properties of the S32DS project. The table below describes the various options available on the **Tools Settings** tab of the **Properties** window. Select **Properties** > **C/C++ Build** > **Settings**, and then select the **Cross Settings** element in the settings tree to display the settings.

Note: The actual list of settings varies depending on the build configuration selected in the **Configuration** list.

Table 8: Tool Settings - Cross Settings

Option	Description
Prefix	Tool chain prefix used to call GNU Tools for ARM Embedded Processors. Default: `powerpc-eabivle-` **Note:** For more information about default values, Restoring Build Properties.
Path	Use to specify the path to GNU Tools for Power Architecture Embedded Processors. Default: `${VLE_TOOLCHAIN_DIR}/bin`
Suffix	Tool chain suffix used to call GNU Tools for Power Architecture Embedded Processors. By default, this field is empty.
C compiler	Use to specify C compiler name. Default: `gcc`
C++ compiler	Use to specify C++ compiler name. Default: `g++`
Hex/Bin converter	Use to specify the utility that copies and translates object files. The gnu **objcopy** utility copies the contents of an object file to another. Default: `objcopy`
Listing generator	Use to specify the utility that displays information about one or more object files. The options control what particular information to display. Default: `objdump`
Size command	Use to specify the utility that processes one or more ELF files. Their output are the dimensions (in bytes) of the text, data and uninitialized sections, and their total. Default: `size`
Build command	Use to specify the utility that automatically builds executable programs and libraries from source code. Default: `make`
Remove command	Use to specify a command to remove executable programs, libraries and object files. Default: `rm -rf`
Create flash image	Select to create flash image using defined Hex/Bin converter utility. By default this check box is cleared.
Create extended listing	Select to create listing using defined Listing generator utility. By default this check box is cleared.

Option	Description

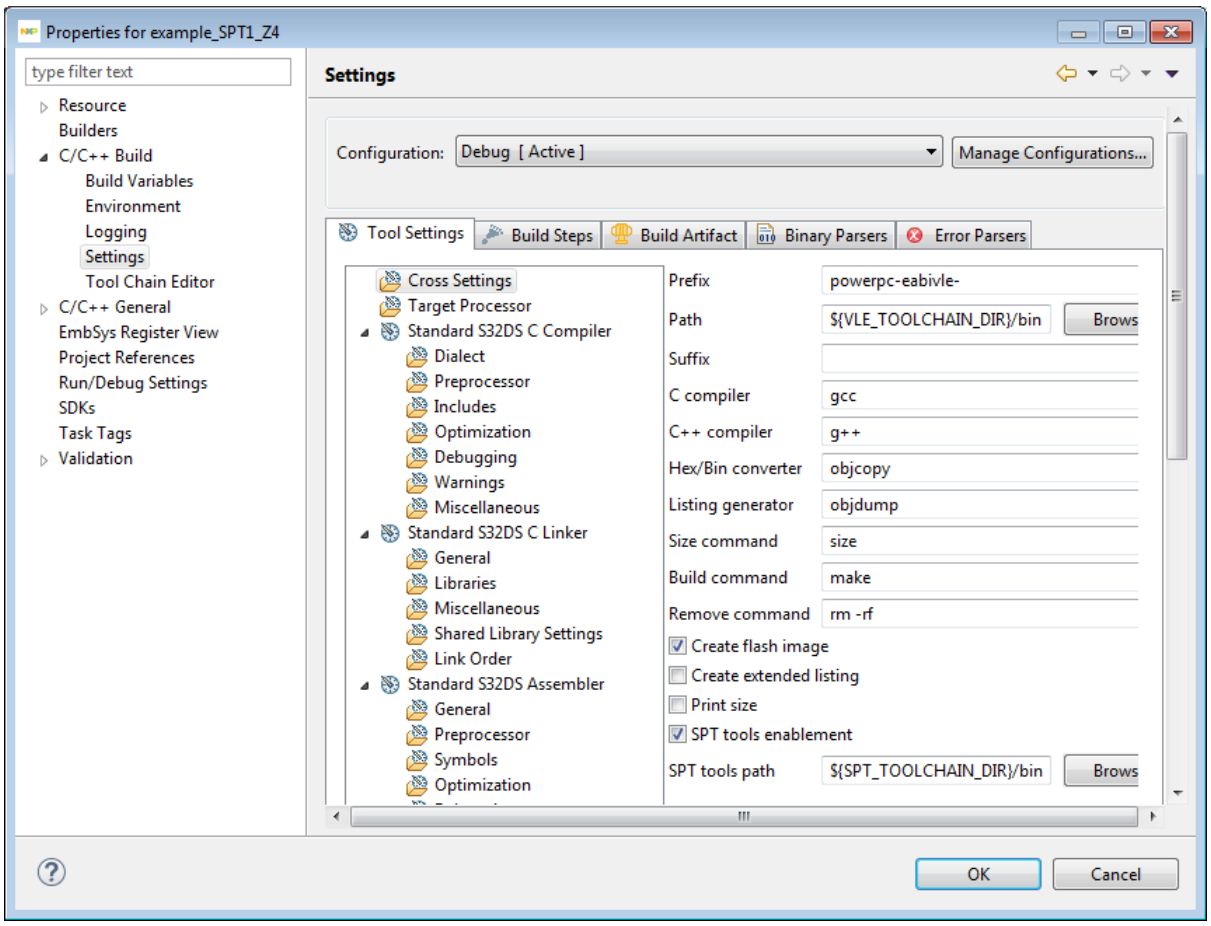
Print size	When selected, S32DS Power v2017.R1 will call the size utility to obtain the details about the produced ELF file. A sample build output:
	`Invoking: Standard S32DS Print Size`
	`powerpc-eabivle-size --format=berkeley`
	` example_SPT1_Z4.elf`
	`text data bss dec hex filename`
	`7249 56 4388 11693 2dad example_SPT1_Z4.elf`
	`Finished building: example_SPT1_Z4.siz`
	By default this check box is cleared.

SPT tools enablement	Check to enable **SPT Assembler** and **SPT disassembler** tools settings (for more information, see SPT Assembler and SPT Disassembler sections). By default this check box is selected.
SPT tools path	Path to SPT toolchain. Default: `${SPT_TOOLCHAIN_DIR}/bin` By default this variable translates to `${eclipse_home}../S32DS/SPT`, where `${eclipse_home}` refers to `eclipse` folder under S32DS Power v2017.R1 program folder. **Note:** This field activates only if **SPT tools enablement** is selected.

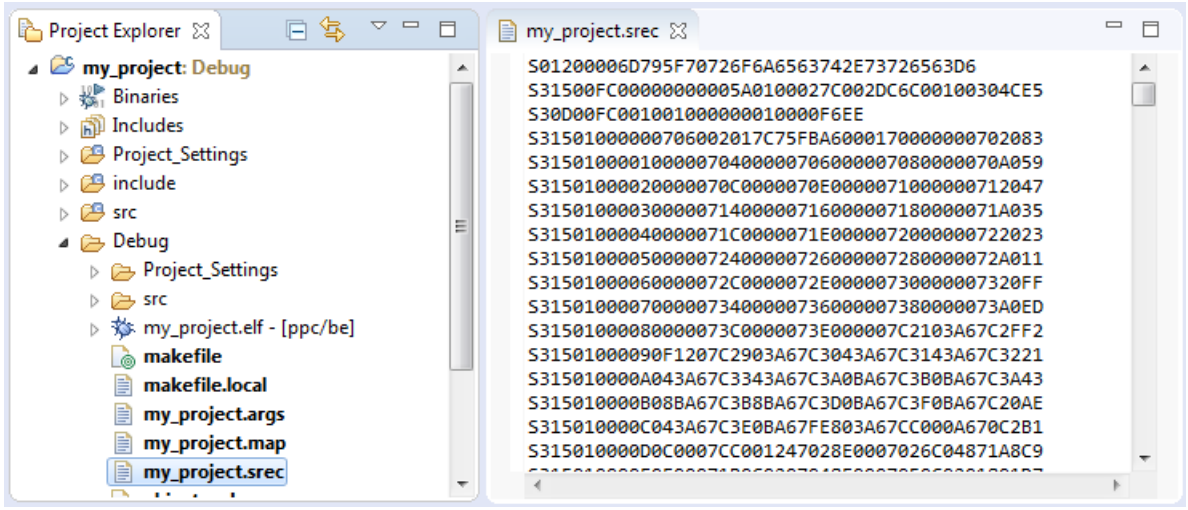
Generate S-record image

This section describes how to generate S-record image. To generate S-record image, perform the following steps:

1. Go to the project **Properties** > **C/C++ Build** > **Settings** > **Cross Settings** and enable **Create flash image** option.



2. Press **Apply** button and **Standard S32DS Create Flash Image** settings should appear in the list.
3. Select **Standard S32DS Create Flash Image** > **General** settings. The **Motorola S-record** value is selected by default from the **Output file format (-O)** list. Press **Apply** and **OK** buttons.
4. Rebuild the project and check **<project name>.srec** file in the **Debug** output directory.



Target Processor

This section describes how to set target processor options in the build properties for Power Architecture project.

Use the **Target Processor** panel to specify the processor family for the project. The properties specified on this page are also used by the build tools (compiler, linker, and assembler).

The table below lists and describes the various options available on the **Target Processor** panel.

Table 9: Tool Settings - Target Processor

Option	Description
Other target flags	Use to specify the additional options according to the compiler documentation. By default, this field is empty.
Target processor	Use to specify the target processor name. The compiler can take advantage of the extra instructions that the selected processor provides and optimize the code to run on a specific processor. The inline assembler might display error messages or warnings if it assembles some processor-specific instructions for the wrong target processor. Following options available: • **generate code for E200Z0**, • **generate code for E200Z2**, • **generate code for E200Z3**, • **generate code for E200Z4**, • **generate code for E200Z7**. Default: **generate code for E200Z2**.
Allow symbolic names for registers	Use to enable (or disable) symbolic names for registers. Default: **on**
Use hardware floating point	Use to enable (or disable) floating-point hardware (or hardware emulation). Basic floating point instructions are supported on some cores. Either scalar or vector floating point options are supported. Default: **off**

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Libraries support</td><td>
Use to specify support of the library, language and I/O mode.
The value format is:
<standard library> <language> <I/O mode>
where:
• standard library – standard library to be linked to project:·
  • ewl - c99 compliant Embedded Warrior Library,
  • ewl_nano - a lightweight version of Embedded Warrior Library,
  • newlib - standard C/C++ library that is included with our toolchain,
  • newlib_nano - a lightweight version of the standard C/C++ library.
• language - language used for project:
  • C,
  • C++
• I/O mode - I/O modes used for project:
  • Debugger Console - configures how the GCC library deals with the console (e.g. printf() or puts()), the library uses a virtual connection with the debugger (also known as “semihosting”),
  • no I/O
Default: None
The options available are: None and values listed in the table:
<table>
<tr><th>Libraries support</th><th>Library</th><th>Lang.</th><th>I/O mode</th></tr>
<tr><td>newlib_nano no I/O</td><td>newlib_nano</td><td></td><td>no I/O</td></tr>
<tr><td>newlib_nano Debugger Console</td><td>newlib_nano</td><td></td><td>Debugger Console</td></tr>
<tr><td>newlib no I/O</td><td>newlib</td><td></td><td>no I/O</td></tr>
<tr><td>newlib Debugger Console</td><td>newlib</td><td></td><td>Debugger Console</td></tr>
<tr><td>ewl_c no I/O</td><td>ewl</td><td>C</td><td>no I/O</td></tr>
<tr><td>ewl_c++ no I/O</td><td>ewl</td><td>C++</td><td>no I/O</td></tr>
<tr><td>ewl_c Debugger Console</td><td>ewl</td><td>C</td><td>Debugger Console</td></tr>
<tr><td>ewl_c++ Debugger Console</td><td>ewl</td><td>C++</td><td>Debugger Console</td></tr>
<tr><td>ewl_nano_c no I/O</td><td>ewl_nano</td><td>C</td><td>no I/O</td></tr>
<tr><td>ewl_nano_c++ no I/O</td><td>ewl_nano</td><td>C++</td><td>no I/O</td></tr>
<tr><td>ewl_nano_c Debugger Console</td><td>ewl_nano</td><td>C</td><td>Debugger Console</td></tr>
<tr><td>ewl_nano_c++ Debugger Console</td><td>ewl_nano</td><td>C++</td><td>Debugger Console</td></tr>
</table>
Only single-core projects support semi-hosting. Multi-core projects do not support semi-hosting (debugger console). Do not select a debugger console library (for example ewl_c Debugger Console or newlib Debugger Console) for your multi-core project.
</td></tr>
</table>

Option	Description
Signal Processing	Use to specify support of Signal Processing Instruction. There are several options for signal processing extensions to the core, either the Lightweight Signal Processing Unit (LSP), the Signal Processing Extension (SPE) or SPE v.2 . Some cores do not support any signal processing instructions. Following options available: • **no**. Some core do not include any type of signal processing extensions • **generate LSP SIMD instructions**. The LSP (version 1) supports a limited number of basic math instructions to speed digital signal processing algorithms • **generate SPE SIMD instructions**. The SPE supports a full range of instructions for digital signal processing algorithms. Support of SPE signal processing instructions version 1.1 • **generate SPE v2 SIMD instructions**. Support of SPE signal processing instructions version 2.1. Also see the Standard S32DS Assembler Panel topic. List of supported SPE2 instructions see in the **Freescale SPE2PIM Rev.2 08/2011** document Default: **no**
Sysroot	Use to specify the logical root directory for headers and libraries. • if the **Libraries support** field contains the **ewl** value then the **Sysroot** value is **-sysroot="="${eclipse_home}../S32DS/e200_ewl2"** • if the **Libraries support** field contains the **newlib** value then the **Sysroot** value is **-sysroot="${VLE_NEWLIB_DIR}/newlib"** By default, this field is empty.

Standard S32DS C Compiler and Standard S32DS C++ Compiler

Standard S32DS C Compiler and Standard S32DS C++ Compiler Panels

This section describes how to specify the C or C++ compiler behavior in the build properties for Power Architecture project. The table below lists and describes the various options available on the **Standard S32DS C Compiler** and **Standard S32DS C++ Compiler** panels.

Table 10: Tool Settings - Standard S32DS C Compiler/ Standard S32DS C++ Compiler panel

	Option	Description
1.	Command	Shows the location of the compiler executable file. • Standard S32DS Power v2017.R1 C Compiler default: **${cross_prefix}${cross_c}${cross_suffix}** • Standard S32DS Power v2017.R1 C++ Compiler default: **${cross_prefix}${cross_cpp}${cross_suffix}**
2.	All options	Shows the actual command line the compiler will be called with. • Standard S32DS Power v2017.R1 C Compiler default: **-O0 -g -Wall -c -fmessage-length=0 -mcpu=e200z2 -mbig -mvle -mregnames -mhard-float** • Standard S32DS Power v2017.R1 C++ Compiler default: **-O2 -g -Wall -c -fmessage-length=0 -mcpu=e200z2 -mbig -mvle -mregnames -mhard-float**

	Option	Description
3.	Command line pattern	Shows the expert settings command line parameters. Default: **${COMMAND} ${FLAGS} -c ${OUTPUT_FLAG} ${OUTPUT_PREFIX} ${OUTPUT} ${INPUTS}**

Dialect

This section describes how to specify the compiler behavior in the build properties for S32DS project.

Language standard option determines the language standard. This option is currently only supported when compiling C or C++.

The compiler can accept several base standards, such as ‘c90’ or ‘c++98’, and GNU dialects of those standards, such as ‘gnu90’ or ‘gnu++98’. When a base standard is specified, the compiler accepts all programs following that standard plus those using GNU extensions that do not contradict it.

For example, **-std=c90** turns off certain features of GCC that are incompatible with ISO C90, such as the **asm** and **typeof** keywords, but not other GNU extensions that do not have a meaning in ISO C90, such as omitting the middle term of a **?:** expression. On the other hand, when a GNU dialect of a standard is specified, all features supported by the compiler are enabled, even when those features change the meaning of the base standard. As a result, some strict-conforming programs may be rejected.

The particular standard is used by **-Wpedantic** to identify which features are GNU extensions given that version of the standard. For example **-std=gnu90 -Wpedantic** warns about C++ style ‘//’ comments, while **-std=gnu99 -Wpedantic** does not.

See Language Standards Supported by GCC, for details of these standard versions.

The table below lists and describes the various options available on the **Dialect** panel.

Table 11: Tool Settings - Standard S32DS C Compiler/ Standard S32DS C++ Compiler - Dialect

	Option	Description
1.	Language standard	Select the programming language or standard to which the compiler should conform. For C compiler possible values are: • ISO C90/ANSI C89 (-std=c90) - Support all ISO C90 programs (certain GNU extensions that conflict with ISO C90 are disabled). Select to compile code written in ANSI standard C. The compiler does not enforce strict standards. For example, code can contain some minor extensions, such as C++ style comments (//), and **$** characters in identifiers, • ISO C99 (-std=c99) - Select to instruct the compiler to enforce stricter adherence to the ANSI/ISO standard. This standard is substantially completely supported, modulo bugs and floating-point issues (mainly but not entirely relating to optional C99 features from Annexes F and G). See gcc.gnu.org for more information, • ISO C11 (-std=c11) – This is the 2011 revision of the ISO C standard. This standard is substantially completely supported, modulo bugs, floating-point issues (mainly but not entirely relating to optional C11 features from Annexes F and G) and the optional Annexes K (Bounds-checking interfaces) and L (Analyzability). For C++ compiler possible values are: • ISO C++98 (-std=c++98) - The 1998 ISO C++ standard plus the 2003 technical corrigendum and some additional defect reports. Same as -ansi for C++ code, • ISO C++11 (-std=c++0x) - The 2011 ISO C++ standard plus amendments, • ISO C++1y (-std=c++1y) - The 2014 ISO C++ standard plus amendments. By default, this field is clear.

	Option	Description
2.	Other dialect flags	Specify additional command line options; type in custom dialect flags that are not otherwise available in the UI. By default, this field is clear.

Preprocessor

This section describes how to specify options controlling the C/C++ preprocessor, which is run on each C/C++ source file before actual compilation. The table below lists and describes the various options available on the **Preprocessor** panel.

Table 12: Tool Settings - Standard S32DS C Compiler/ Standard S32DS C++ Compiler - Preprocessor

	Option	Description
1.	Do not search system directories (-nostdinc)	Check if you do not want the C compiler to search the system directories for header files. The C compiler performs a full search that includes the system directories. By default this checkbox is clear.
2.	Preprocess only (-E)	Check to preprocess source files and not to run the compiler. Nothing is done except preprocessing. By default, this checkbox is clear and the source files are not preprocessed.
3.	Do not search system C++ directories (-nostdinc++) (C++ only)	Check if you do not want the C++ compiler to search the system directories for header files. The C++ compiler performs a full search that includes the system directories. By default this checkbox is clear.

The **Preprocessor** panel for C/C++ Compiler contains **Defined symbols** and **Undefined symbols** fields as well (see Symbols paragraph).

Symbols

This section describes how to specify the C compiler behavior in the build properties for S32DS project. The table below lists and describes the various options available on the **Symbols** panel.

Table 13: Tool Settings - Standard S32DS C Compiler - Symbols

	Option	Description
1.	Defined symbols (-D)	Use to specify the substitution strings that the compiler applies to all the assembly-language modules in the build target. Enter just the string portion of a substitution string. The S32DS Power v2017.R1 prepends the **-D** token to each string that you enter. For example, entering **opt1 x** produces this result on the command line: **-Dopt1 x** **Note:** This option is similar to the **#define** directive, but applies to all assembly-language modules in a build target. By default this checkbox is clear.

	Option	Description
2.	Undefined symbols (-U)	Cancel any previous definition of name, either built in or provided with a **-D** option By default this checkbox is clear.

Includes

This section describes how to specify the directories paths and files paths in the build properties for C/C++ S32DS project. The table below lists and describes the various options available on the **Includes** panel.

Table 14: Tool Settings - Standard S32DS C Compiler/ Standard S32DS C++ Compiler - Includes

	Option	Description
1.	Include paths (-I)	This option adds the directory to the list of directories to be searched for header files. Directories named by -I are searched before the standard system include directories. If the directory is a standard system include directory, the option is ignored to ensure that the default search order for system directories and the special treatment of system headers are not defeated. This option changes the build target's search order of access paths to start with the system paths list. The compiler can search **#include** files in several different ways. You can also set the search order as follows: For include statements of the form **#include"xyz"**, the compiler first searches user paths, then the system paths. For include statements of the form **#include<xyz>**, the compiler searches only system paths. This option is global. By default, this option is set to `"${ProjDirPath}/include"`
2.	Include files (-include)	Use this option to specify the include file search path. Process file as if **#include "file"**appeared as the first line of the primary source file. However, the first directory searched for file is the preprocessor's working directory instead of the directory containing the main source file. If not found there, it is searched for in the remainder of the **#include** "..." search chain as normal. If multiple **-include** options are given, the files are included in the order they appear on the command line. By default this checkbox is clear.

Optimization

This section describes how to control C/C++ compiler optimizations in the build properties for S32DS project. Compiler optimization can be applied in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Without any optimization option, the C/C++ compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you expect from the source code.

Turning on optimization flags makes the C/C++ compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

The C/C++ compiler performs optimization based on the knowledge it has of the program. Compiling multiple files at once to a single output file mode allows the compiler to use information gained from all of the files when compiling each of them.

The table below lists and describes the various options available on the **Optimization** panel.

Table 15: Tool Settings - Standard S32DS C Compiler/ Standard S32DS C++ Compiler - Optimization

	Option	Description
1.	Optimization level	Specify the optimizations that you want the C/C++ compiler to apply to the generated object code: • **None (-O0)** - Disable optimizations. This setting is equivalent to specifying the **-O0** command-line option. The compiler generates unoptimized, linear assembly-language code, reduce compilation time. • **Optimize (-O1)** - The compiler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. This setting is equivalent to specifying the **-O1** command-line option. The compiler omits all target-specific optimizations and generates linear assembly-language code. • **Optimize more (-O2)** - The compiler performs all optimizations (both target-independent and target-specific). GCC performs nearly all supported optimiza-tions that do not involve a space-speed tradeoff. As compared to **-O**, this option increases both compila-tion time and the performance of the genera-ted code. This setting is equivalent to specifying the **-O2** com-mand-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code. • **Optimize most (-O3)** - The compiler performs all the level 2 optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the **-O3** command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level 2 optimizations. • **Optimize size (-Os)** - The compiler performs further optimizations designed to reduce code size. **-Os** enables all **-O2** optimizations that do not typically increase code size. The resulting binary file has a smaller executable code size, as opposed to a faster execution speed. This setting is equivalent to specifying the **-Os** command-line option. **Note:** Default value of optimization level depends not only on com-pilers (C or C++) but on configuration (Debug or Release) as well. Standard S32DS C/C++ Compiler defaults: • for debug configuration: **None (-O0)** • for release configuration: **Optimize most (-O3)**
2.	Other optimization flags	Specifies additional command line options and individual optimization flag that can be turned ON/OFF based on the user requirements. Type in custom optimization flags that are not otherwise available in the UI. By default this checkbox is clear.

	Option	Description
3.	'char' is signed (-fsigned-char) (Standard S32DS C Compiler **only**)	Check to treat char declarations as signed char declarations. This setting is equivalent to **-fsigned-char** command-line option. By default this checkbox is clear.
4.	Function sections (-ffunction-sections)	Check to use function sections. Place each function into its own section in the output file if the target supports arbitrary sections. The name of the function determines the section's name in the output file. Use these options on systems where the linker can perform optimizations to improve locality of reference in the instruction space. Most systems using the ELF object format have linkers with such optimizations. Only use these options when there are significant benefits from doing so. This setting is equivalent to **-ffunction-sections** command-line option. By default, this check box is selected.
5.	Data sections (-fdata-sections)	Check to use short data sections. Place each data item into its own section in the output file if the target supports arbitrary sections. The name of the data item determines the section's name in the output file. Use these options on systems where the linker can perform optimizations to improve locality of reference in the instruction space. Most systems using the ELF object format have linkers with such optimizations. Only use these options when there are significant benefits from doing so. When you specify these options, the assembler and linker create larger object and executable files and are also slower. You cannot use **gprof** on all systems if you specify this option, and you may have problems with debugging if you specify both this option and **-g**. This setting is equivalent to **-ffunction-sections** command-line option. By default, this check box is selected.

	Option	Description
6.	No common uninitialized (-fno-common)	In C code, controls the placement of uninitialized global variables. Unix C compilers have traditionally permitted multiple definitions of such variables in different compilation units by placing the variables in a common block. This is the behavior specified by **-fcommon**, and is the default for GCC on most targets. On the other hand, this behavior is not required by ISO C, and on some targets may carry a speed or code size penalty on variable references. The **-fno-common** option specifies that the compiler should 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 **-fcommon** instead. Compiling with **-fno-common** 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. This setting is equivalent to **-fno-common** command-line option. By default this checkbox is clear.
7.	Do not inline functions (-fno-inline-functions)	Do not consider any functions for inlining, even if they are not declared inline. This setting is equivalent to **-fno-inline-functions** command-line option. By default this checkbox is clear.
8.	Assume freestanding environment (-ffreestanding)	Assert that compilation targets a freestanding environment. This implies **-fno-builtin**. A freestanding environment is one in which the standard library may not exist, and program startup may not necessarily be at main. The most obvious example is an OS kernel. This is equivalent to **-fno-hosted**. By default this checkbox is clear.
9.	Disable builtin (-fno-builtin)	Don't recognize built-in functions that do not begin with **__builtin_** as prefix. This setting is equivalent to **-fno-builtin** command-line option. By default this checkbox is clear.
10.	Single precision constants (-fsingle-precision-constant)	Check to enable single precision constants. Treat floating-point constants as single precision instead of implicitly converting them to double-precision constants. This setting is equivalent to **-single-precision-constant** command-line option. By default this checkbox is clear.

	Option	Description
11.	Link-time optimizer (-flto)	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. This setting is equivalent to **-flto** command-line option. By default this checkbox is clear.
12.	Disable loop invariant move (-fno-move-loop-invariants)	Disable the loop invariant motion pass in the RTL loop optimizer. Enabled at level **-O1**. This setting is equivalent to **-fno-move-loop-invariants** command-line option. By default this checkbox is clear.

Debugging

This section describes how to control debugging in the build properties for the C/C++ S32DS project.

The table below lists and describes the various options available on the **Debugging** panel.

Table 16: Tool Settings - Standard S32DS C Compiler/ Standard S32DS C++ Compiler - Debugging

	Option	Description
1.	Debug Level	Specify the debug levels: • **None** - Level 0 produces no debug information at all, • **Minimal (-g1)** - Level 1 produces minimal information, enough for making backtraces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, and line number tables, but no information about local variables, • **Default (-g)** - The default level is 2. The compiler generates DWARF 1.x conforming debugging information, • **Maximum (-g3)** - Level 3 includes extra information, such as all the macro definitions present in the program. So the compiler provides maximum debugging support. Default: **Maximum (-g3)**
2.	Other debugging flags	Specify additional command line options; type in custom debugging flags that are not otherwise available in the UI. By default, this field is clear.
3.	Generate gcov information (-ftest-coverage -fprofile-arcs)	Select to enable Profile Code Coverage in your application. Remember to enable this option in both the **Compiler** > **Miscellaneous** and **Linker** > **General**. Then rebuild your project and run Code Coverage again. By default, this field is clear.

	Option	Description
4.	Debug format	Specify the debug formats for the compiler. Use it if you want to control for certain whether to generate the extra information. • **Toolchain default**, • **Gdb** • **stabs** - Generates STABS-conforming debugging information, • **stabs+**, • **dwarf-2** - Generates DWARF 2.x-conforming debugging information, • **dwarf-3** - Generates DWARF 3.x-conforming debugging information, • **dwarf-4** - Generates DWARF 4.x-conforming debugging information. Default: **Toolchain default.**

Note:

Only **Other debugging flags** field is available. If a project contains some debug flags set in **Other options** then these options will present in build system but not available for changing or removing. The user shall edit .cproject file to move this flags to the **Other debugging flags** field.

For example, the line in **.cproject** file:

```
<option id="com.freescale.s32ds.cross.gnu.tool.c.compiler.option.debugging.other.842367527"
superClass="com.freescale.s32ds.cross.gnu.tool.c.compiler.option.debugging.other"
value="test_opt"
valueType="string"/>
```

should be changed to

```
<option id="gnu.c.compiler.option.debugging.other.1735145584"
superClass="gnu.c.compiler.option.debugging.other"
value="test_opt"
valueType="string"/>
```

The options will be moved to the **Other debugging flags** field.

Warnings

This section describes how to control C/C++ compiler warnings in the build properties for S32DS project.

The table below lists and describes the various options available on the **Warnings** panel.

Table 17: Tool Settings - Standard S32DS C Compiler/ Standard S32DS C++ Compiler - Warnings

	Option	Description
1.	Check syntax only (-fsyntax-only)	Check to check the code for syntax errors, but do not do anything beyond that. By default, this field is clear.
2.	Pedantic (-pedantic)	Check to issue all the warnings demanded by strict ISO C and ISO C ++; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. For ISO C, follows the version of the ISO C standard specified by any **-std** option used. By default, this field is clear.

	Option	Description
3.	Pedantic warnings as errors (-pedantic-errors)	Like pedantic, except that errors are produced rather than warnings. By default, this field is clear.
4.	Inhibit all warnings (-w)	Check to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros. By default, this field is clear.
5.	All warnings (-Wall)	Check to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros. This also enables some language-specific warnings. Default: on
6.	Extra warnings (-Wextra)	Check to enable some extra warning flags that are not enabled by **-Wall**. By default, this field is clear.
7.	Warnings as errors (-Werror)	Check to make all warnings into hard errors. Source code which triggers warnings will be rejected. The specifier for a warning is appended. By default, this field is clear.
8.	Implicit conversions warnings (-Wconversion)	Check to warn for implicit conversions that may alter a value. This includes conversions between real and integer, like **abs (x)** when x is double; conversions between signed and unsigned, like unsigned **ui = -1**; and conversions to smaller types, like **sqrtf (M_PI)**. Do not warn for explicit casts like **abs ((int) x)** and **ui = (unsigned) -1**, or if the value is not changed by the conversion like in **abs (2.0)**. Warnings about conversions between signed and unsigned integers can be disabled by using **-Wno-sign-conversion**. By default, this field is clear.
9.	Warn on uninitialized variables (-Wuninitialized)	Check to warn if an automatic variable is used without first being initialized or if a variable may be clobbered by a **setjmp** call. In C++, warn if a non-static reference or non-static const member appears in a class without constructors. To warn about code that uses the uninitialized value of the variable in its own initializer, use the **-Winit-self** option. These warnings occur for individual uninitialized or clobbered elements of structure, union or array variables as well as for variables that are uninitialized or clobbered as a whole. They do not occur for variables or elements declared volatile. Because these warnings depend on optimization, the exact variables or elements for which there are warnings depend on the precise optimization options and version of GCC used. Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by data flow analysis before the warnings are printed. By default, this field is clear.

	Option	Description
10.	Warn on various unused elements (-Wunused)	Check to warn if various constructs are unused (parameters, local variables, functions, labels, typedef locally defined, function parameters). By default, this field is clear.
11.	Warn if padding is included (-Wpadded)	Check to warn if padding is included in a structure, either to align an element of the structure or to align the whole structure. Sometimes when this happens it is possible to rearrange the fields of the structure to reduce the padding and so make the structure smaller. By default, this field is clear.
12.	Warn if floats are compared as equal (-Wfloat-equal)	Check to warn if floating-point values are used in equality comparisons. The idea behind this is that sometimes it is convenient (for the programmer) to consider floating-point values as approximations to infinitely precise real numbers. If you are doing this, then you need to compute (by analyzing the code, or in some other way) the maximum or likely maximum error that the computation introduces, and allow for it when performing comparisons (and when producing output, but that's a different problem). In particular, instead of testing for equality, you should check to see whether the two values have ranges that overlap; and this is done with the relational operators, so equality comparisons are probably mistaken. By default, this field is clear.
13.	Warn if shadowed variable (-Wshadow)	Check to warn 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 that in C++, the compiler warns if a local variable shadows an explicit typedef, but not if it shadows a struct/class/enum. By default, this field is clear.
14.	Warn if pointer arithmetic (-Wpointer-arith)	Check to warn about anything that depends on the **sizeof** a function type or of void. GNU C assigns these types a size of 1, for convenience in calculations with void ***** pointers and pointers to functions. In C++, warn also when an arithmetic operation involves **NULL**. This warning is also enabled by **-Wpedantic**. By default, this field is clear.
15.	Warn if suspicious logical ops (-Wlogical-op)	Check to warn 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. By default, this field is clear.
16.	Warn if struct is returned (-Wagreggate-return)	Check to warn if any functions that return structures or unions are defined or called. (In languages where you can return an array, this also elicits a warning). By default, this field is clear.

	Option	Description
17.	Warn on undeclared global function (-Wmissing-declaration)	Check to warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files. In C, no warnings are issued for functions with previous non-prototype declarations; use **-Wmissing-prototype** to detect missing prototypes. In C++, no warnings are issued for function templates, or for inline functions, or for functions in anonymous namespaces. By default, this field is clear.
18.	Other warning flags	Specifies additional command line options and individual warning flag that can be turned ON/OFF based on the user requirements. Type in custom warning flags that are not otherwise available in the UI. By default, this field is clear.

Miscellaneous

This section describes how to specify miscellaneous C/C++ compiler options.

The table below lists and describes the various options available on the **Miscellaneous** panel.

Table 18: Tool Settings - Standard S32DS C Compiler/ Standard S32DS C++ Compiler - Miscellaneous

	Option	Description
1.	Other flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI. Default: **-c -fmessage-length=0**
2.	Verbose (-v)	Check to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the **-v** command-line option. By default this checkbox is clear. The S32DS Power v2017.R1 displays just error messages that the compiler emits. The IDE suppresses warning and informational messages.

	Option	Description
3.	Support ANSI programs (-ansi) (Standard S32DS C Compiler **only**)	Check this option if you want the assembler to operate in strict ANSI mode. In this mode, the compiler strictly applies the rules of the ANSI/ ISO specification to all input files. This 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 **asm** and **typeof** keywords, and predefined macros such as **unix** and **vax** 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 '//' comments as well as the inline keyword. The alternate keywords **__asm__**, **__extension__**, **__inline__** and **__typeof__** continue to work despite **-ansi**. You would not want to use them in an ISO C program, of course, but it is useful to put them in header files that might be included in compilations done with **-ansi**. Alternate predefined macros such as **__unix__** and **__vax__** are also available, with or without **-ansi**. The **-ansi** option does not cause non-ISO programs to be rejected gratuitously. For that, **-Wpedantic** is required in addition to **-ansi**. The macro **__STRICT_ANSI__** is predefined when the **-ansi** option is used. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ISO standard doesn't call for; this is to avoid interfering with any programs that might use these names for other things. Functions that are normally built in but do not have semantics defined by ISO C (such as **alloca** and **ffs**) are not built-in functions when **-ansi** is used. This setting is equivalent to specifying the **-ansi** command-line option. The compiler issues a warning for each ANSI/ISO extension it finds. By default this checkbox is clear. The assembler does not operate in strict ANSI mode
4.	Position Independent Code (-fPIC)	If supported for the target, emit position-independent code, suitable for dynamic linking and avoiding any limit on the size of the global offset table. Position-independent code requires special support, and therefore works only on certain machines. When this flag is set, the macros **__pic__** and **__PIC__** are defined to 2. By default this checkbox is clear.
5.	Save temporary files (--save-temps)	Enables you to save temporary intermediate files and place them in the current directory and name them based on the source file. Use with caution! By default this checkbox is clear.
6.	Generates assembler listing (-Wa, -adhlns="$@.lst")	Enables the assembler to create a listing file as it compiles assembly language into object code. By default this checkbox is clear.

Supporting of specifications

This section describes features of the compiler which support documented intrinsic requirements.

PowerISA 2.07b

Following instructions are marked as VLE according to PowerISA 2.07b:

evlddepx evstddepx e sc se rfgi sld sld. srad srad. srd srd. stbcx. stbdx stddx stfddx sthcx. sthdx stvebx stvehx stvepx stvepxl stvewx stwdx vcmpbfp. subfo subfo. subfmeo subfmeo.

EFP2 rev1.4

Changes in accordance with EFP2 rev1.4:

- moved opcodes for:

  efdcfsid efdcfuid efdctsidz efdctuidz
- added instructions:

  evfssqrt evfscfh evfscth evfsmax evfsmin evfsaddsub evfssubadd evfssum evfsdiff evfssumdiff evfsdiffsum evfsaddx evfssubx evfsaddsubx evfssubaddx evfsmulx evfsmule evfsmulo efsmax efsmin efdmadd efdmsub efdnmadd efdnmsub efssqrt efscfh efscth efdmax efdmin efdsqrt efdcfh efdcth

SPE2PIM Rev.1.0-2

Added mapping for SPE2 instructions in accordance with SPE2PIM Rev.1.0-2:

evdotphsssi evdotphsssia evdotpwsssi evdotpwsssia

Engineering Bulletin EB689 Rev. 0, 2/2008

Added instructions from Engineering Bulletin EB689 Rev. 0, 2/2008 for e200z3 and e200z6 cores:

evfsmadd evfsmsub evfsnmadd evfsnmsub efsmadd efsmsub efsnmadd efsnmsub

BookE support

This section describes features of the compiler which support BookE code version.

Mixed BookE and VLE code assembly and linking

Codegen version	Compiler features
BookE	Use -mno-vle key to force BookE codegen, even if -mvle key is used

Mixed VLE and BookE	For mixed VLE and BookE code corresponding linker script shall be provided, for example:
	`.text_booke : INPUT_SECTION_FLAGS (!SHF_PPC_VLE) *(.text*)> text_e`
	`.text_booke : INPUT_SECTION_FLAGS (!SHF_PPC_VLE) *(.text*)> m_text`
	• first line will force pick non-VLE .text sections
	• second line will force pick VLE .text sections

BookE to VLE translation

Use **-Wa,-ppc_asm_to_vle** option to enable translation for assembler file. Option requires **-mvle** option.

Following BookE instructions can be translated:

addi addic addic. addis andi. andis. b bc bcl bctr bctrl bdnz bdnzl bdz bdzl beq beql bf bfl bge bgel bgt bgtl bl ble blel blr blt bltl bne bnel bng bngl bnl bnll bns bnsl bnu bnul bso bsol bt btl bun bunl clrlslwi clrlwi clrrwi cmpi cmpli cmplwi cmpwi crand crandc crclr creqv crmove crnand crnor crnot cror crorc crset crxor extlwi extrwi inslwi insrwi isync lbz lbzu lha lhau lhz lhzu li lis lmw lwz lwzu mcrf mfdec mtdec mulli nop ori oris rfi rlwimi rlwinm rlwnm rotlw rotlw. rotlwi rotrwi sc slwi srwi stb stbu sth sthu stmw stw stwu subfic subi subic subic. subis xnop xori.

Following VLE-compatible instructions can be translated to shorter form:

mr or.

Devices with BookE support

In the S32DS Power v2017.R1 following devices have the MMU and VLE/BookE instruction set support:

- MPC5643L
- MPC5646R
- MPC5646C/B (only for e200z4d)
- MPC5777C.

You can import examples which used VLE and BooKE code (see the **S32 Design Studio for Power Architecture, Version 2017.R1 Common Features Guide** > **Import example project** topic).

Standard S32DS C Linker and Standard S32DS C++ Linker

Standard S32DS C Linker and Standard S32DS C++ Linker Panel

This section describes how to specify the C/C++ linker behavior in the build properties for Power Architecture project. The table below lists and describes the various options available on the Standard S32DS C Linker and Standard S32DS C++ Linker panels.

Table 19: Tool Settings - Standard S32DS C Linker/ Standard S32DS C++ Linker panel

	Option	Description
1.	Command	Shows the location of the linker executable file. - Standard S32DS Power v2017.R1 C Linker default: **gcc** - Standard S32DS Power v2017.R1 C++ Linker default: **g++**
2.	All options	Shows the actual command line the linker will be called with. Default: **-Wl,-Map,"<project name>.map" -mcpu=e200z2 -mhard-float**
3.	Command line patterns	Shows the expert settings command line parameters. Default: **${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX} ${OUTPUT} ${INPUTS}**

General

This section describes how to specify the C/C++ linker behavior in the build properties for S32DS project. The table below lists and describes the various options available on the **General** panel.

Table 20: Tool Settings - Standard S32DS C Linker/ Standard S32DS C++ Linker - General

	Option	Description
1.	Do not use standard start files (-nostartfiles)	This option passes the **-nostartfiles** argument to the C/C++ linker file. It does not allow the use of the standard system startup files when linking. The standard system libraries are used normally, unless **-nostdlib** or **-nodefaultlibs** is used By default this checkbox is clear.
2.	Do not use default libraries (-nodefaultlibs)	This option passes the **-nodefaultlibs** argument to the C/C++ linker file. It does not allow the use of the default system libraries when linking. Only the libraries you specify are passed to the linker, and options specifying linkage of the system libraries, such as **-static-libgcc** or **-shared-libgcc**, are ignored By default this checkbox is clear.
3.	No startup or default libs (-nostdlib)	This option passes the **-nostdlib** argument to the C/C++ linker file. It does not allow the use of startup system files or libraries when linking. No startup files and only the libraries you specify are passed to the linker, and options specifying linkage of the system libraries, such as **-static-libgcc** or **-shared-libgcc**, are ignored. The **libgcc.a** library standard bypasses **–nostdlib** and **–nodefaultlibs**. By default this checkbox is clear.
4.	Omit all symbols information (-s)	This option passes the **-s** argument to the C/C++ linker file. This option omits all symbol information and remove all symbol table and relocation information from the executable By default this checkbox is clear.
5.	No shared libraries (-static) (Standard S32DS C Linker **only**)	This option passes the **-static** argument to the C linker file. It does not allow the use of the shared libraries on systems that support dynamic linking By default this checkbox is clear.
6.	Script files (-T)	This option passes the **-T** argument to the C/C++ linker file. This option is supported by most systems using the GNU linker. On some targets, such as bare-board targets without an operating system, the **-T** option may be required when linking to avoid references to undefined symbols For A53 SoC, this parameter is set to: `"${ProjDirPath}/Project_Settings/Linker_Files/S32V_CA53_ram.ld"` .

Libraries

This section describes how to specify libraries behavior in the build properties for S32DS project. The table below lists and describes the various options available on the **Libraries** panel.

Table 21: Tool Settings - Standard S32DS C Linker/ Standard S32DS C++ Linker - Libraries

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

Miscellaneous

This section describes how to specify miscellaneous C/C++ linker options. The table below lists and describes the various options available on the Miscellaneous panel.

Table 22: Tool Settings - Standard S32DS C Linker/ Standard S32DS C++ Linker - Miscellaneous

	Option	Description
1.	Linker flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI. By default, this field is empty.
2.	Other options **(-Xlinker [option])**	Specify additional command line options; type in custom flags that are not otherwise available in the UI. By default, this field is empty.
3.	Other objects	This option lists paths that the linker searches for objects. The linker searches the paths in the order shown in this list. By default, this field is empty.
4.	Generate map	This option specifies the map filename. Default: **$ {BuildArtifactFileBaseName}.map**
5.	Cross reference **(-Xlinker --cref)**	Check this option to instruct the linker to list cross-reference information on symbols. This includes where the symbols were defined and where they were used, both inside and outside macros. By default this checkbox is clear.
6.	Print link map **(-Xlinker --printf-map)**	Check this option to instruct the linker to print the map file. By default this checkbox is clear.

	Option	Description
7.	Remove unused sections **(-Xlinker --gc-sections)**	This option passes the **-Xlinker --gc-sections** argument to the linker file. It removes the unused sections. By default this checkbox is selected.
8.	Print removed sections **(-Xlinker --print-gc-sections)**	This option passes the **-Xlinker --print-gc-sections** argument to the linker file. It prints the removed sections. By default this checkbox is clear.

Standard S32DS Assembler

Standard S32DS Assembler Panel

This section describes how to specify the Assembler behavior in the build properties for Power Architecture C/C++ project. The table below lists and describes the various options available on the Standard S32DS Assembler panel.

Table 23: Tool Settings - Standard S32DS Assembler pane

	Option	Description
1.	Command	Shows the location of the assembler executable file. Default: **${cross_prefix}${cross_c}${cross_suffix}**
2.	All options	Shows the actual command line the assembler will be called with. Default: **-x assembler -O0 -g -mcpu=e200z2 -mbig -mvle -mregnames**
3.	Command line pattern	Shows the expert settings command line parameters. Default: **${COMMAND} ${FLAGS} -c ${OUTPUT_FLAG} ${OUTPUT_PREFIX} ${OUTPUT} ${INPUTS}**

Use **-mspe2** option to enable SPE2 instructions support in assembler. SPE2 instructions can be used by assembler inline syntax or pure assembler. For example:

```
__asm("evdotpwcssi 0, 1, 2");
```

Also see the Target Processor topic (the **Signal Processing** option description).

List of supported SPE2 instructions can be found in the **Freescale SPE2PIM Rev.2 08/2011** document.

General

This section describes how to set assembler options in the build properties for C/C++ S32DS project. The table below lists and describes the various options available on the **General** panel.

Table 24: Tool Settings - Standard S32DS Assembler - General

	Option	Description
1.	Assembler flags	Specify the flags that need to be passed with the assembler By default this checkbox is clear.

	Option	Description
2.	Include paths (-I)	This option changes the build target's search order of access paths to start with the system paths list. The compiler can search **#include** files in several different ways. You can also set the search order as follows: • for include statements of the form **#include"xyz"**, the compiler first searches user paths, then the system paths • for include statements of the form **#include<xyz>**, the compiler searches only system paths. This option is global. By default, this parameter is set to `"${ProjDirPath}/include"`.
3.	Suppress warnings (-W)	This enables some extra warning flags that are not enabled by **-Wall**. By default this checkbox is clear.
4.	Announce version (-v)	Check this option if you want the S32DS Power v2017.R1 to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the **-v** command-line option. The IDE displays just error messages that the compiler emits. The IDE suppresses warning and informational messages. By default this checkbox is clear.

Preprocessor

This section describes how to specify assembler options in the build properties for S32DS C/C++ project. The table below lists and describes the various options available on the **Preprocessor** panel.

Table 25: Tool Settings - Standard S32DS Assembler - Preprocessor

	Option	Description
1.	Use preprocessor	Check this option to use the preprocessor for the assembler. By default, this check box is selected.
2.	Do not search system directories (-nostdinc)	Check if you do not want the assembler to search the system directories. The assembler performs a full search that includes the system directories. By default this checkbox is clear.
3.	Preprocess only (-E)	Check if you want the assembler to preprocess source files and not to run the compiler. Nothing is done except preprocessing. By default, this checkbox is clear and the source files are not preprocessed.

Symbols

This section describes how to set assembler options in the build properties for S32DS C/C++ project. The table below lists and describes the various options available on the **Symbols** panel.

Table 26: Tool Settings - Standard S32DS Assembler - Symbols

	Option	Description
1.	Defined symbols (-D)	Use this option to specify the substitution strings that the assembler applies to all the assembly-language modules in the build target. Enter just the string portion of a substitution string. The S32DS Power v2017.R1 prepends the **-D** token to each string that you enter. For example, entering **opt1 x** produces this result on the command line: **-Dopt1 x** **Note:** This option is similar to the DEFINE directive, but applies to all assembly-language modules in a build target By default this checkbox is clear.
2.	Undefined symbols (-U)	Undefines the substitution strings you specify in this panel By default this checkbox is clear.

Optimization

This section describes how to control optimizations in the build properties for S32DS C/C++ project. The table below lists and describes the various options available on the **Optimization** panel.

Table 27: Tool Settings - Standard S32DS Assembler - Optimization

	Option	Description
1.	Optimization level	Specify the optimizations that you want the assembler to apply to the generated object code: • **None (-O0)** - Disable optimizations. This setting is equivalent to specifying the - **O0** command-line option. The assembler generates unoptimized, linear assembly-language code, reduce compilation time. • **Optimize (-O1)** - The assembler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. Optimizing takes somewhat more time, and a lot more memory for a large function. This setting is equivalent to specifying the **-O1** command-line option. The assembler omits all target-specific optimizations and generates linear assembly-language code. • **Optimize more (-O2)** - The assembler performs all optimizations (both target-independent and target-specific). GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to **-O**, this option increases both compilation time and the performance of the generated code. This setting is equivalent to specifying the **-O2** command-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code. • **Optimize most (-O3)** - The assembler performs all the level 2 optimizations then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the **-O3** command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level 2 optimizations. • **Optimize size (-Os)** - The assembler performs further optimizations designed to reduce code size. **-Os** enables all **-O2** optimizations that do not typically increase code size. The resulting binary file has a smaller executable code size, as opposed to a faster execution speed. This setting is equivalent to specifying the **-Os** command-line option. Default: **None (-O0)**
2.	Other optimization flags	Specifies additional command line options and individual optimization flag that can be turned ON/OFF based on the user requirements. Type in custom optimization flags that are not otherwise available in the UI. By default this checkbox is clear.

Debugging

This section describes how to control debugging in the build properties for C/C++ S32DS project. The table below lists and describes the various options available on the **Debugging** panel.

Table 28: Tool Settings - Standard S32DS Assembler - Debugging

	Option	Description
1.	Debug Level	Specify the debug levels: • **None** - Level 0 produces no debug information at all, • **Minimal (-g1)** - Level 1 produces minimal information, enough for making backtraces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, and line number tables, but no information about local variables, • **Default (-g)** - The default level is 2. The compiler generates DWARF 1.x conforming debugging information, • **Maximum (-g3)** - Level 3 includes extra information, such as all the macro definitions present in the program. So the compiler provides maximum debugging support. Default: **Maximum (-g3)**
2.	Other debugging flags	Specify additional command line options; type in custom debugging flags that are not otherwise available in the UI. By default, this field is clear.

Standard S32DS C Preprocessor and Standard S32DS C++ Preprocessor

Standard S32DS C Preprocessor and Standard S32DS C++ Preprocessor Panels

This section describes how to specify the C/C++ preprocessor behavior in the build properties for S32DS project. The table below lists and describes the various options available on the **Standard S32DS C Preprocessor** and **Standard S32DS C++ Preprocessor** panels.

Table 29: Tool Settings - Standard S32DS C Preprocessor/ Standard S32DS C++ Preprocessor pane

	Option	Description
1.	Command	Shows the location of the preprocessor executable file. • Standard S32DS C Preprocessor default: **${cross_prefix}${cross_c}${cross_suffix}** • Standard S32DS C++ Preprocessor default: **${cross_prefix}${cross_cpp}${cross_suffix}**
2.	All options	Shows the actual command line the preprocessor will be called with. Default: **-E**
3.	Command line patterns	Shows the expert settings command line parameters. • Standard S32DS C Preprocessor default: **${COMMAND} ${FLAGS} ${INPUTS}** • Standard S32DS C++ Preprocessor default: **${COMMAND} ${FLAGS} -E ${INPUTS}**

Settings

This section describes how to specify the preprocessor settings in the build properties for S32DS project. The table below lists and describes the various options available on the **Settings** panel.

Table 30: Tool Settings - Standard S32DS C Preprocessor/ Standard S32DS C++ Preprocessor - Settings

	Option	Description
1.	Handle Directives Only fdirectives-only)	Check to specify the **-fdirectives-only** command to the C/C++ preprocessor to handle only directives, but do not expand macros. The option's behavior depends on the **-E** and **-fpreprocessed** options. By default this checkbox is clear.
2.	Print Header File Names (-H)	Check to specify the **-H** command to the C/C++ preprocessor to print the name of each header file used, in addition to other normal activities. Precompiled header files are also printed, even if they are found to be invalid. By default this checkbox is clear.

Standard S32DS Disassembler

Standard S32DS Disassembler Panel

This section describes how to specify the disassembler behavior in the build properties for C/C++ S32DS project. The table below lists and describes the various options available on the **Standard S32DS Disassembler** panel.

Table 31: Tool Settings - Standard S32DS Disassembler pane

	Option	Description
1.	Command	Shows the location of the disassembler executable file. Default: **${cross_prefix}objdump${cross_suffix}**
2.	All options	Shows the actual command line the disassembler will be called with. Default: **-d -S -x**
3.	Command line pattern	Shows the expert settings command line parameters. Default: **${COMMAND} ${FLAGS} ${INPUTS}**

Settings

This section describes how to specify the disassembler settings in the build properties for S32DS project. The table below lists and describes the various options available on the **Settings** panel.

Table 32: Tool Settings - Standard S32DS Disassembler - Settings

	Option	Description
1.	Disassemble All Section Content (including debug information) (-D)	Check to specify the **-D** command to the disassembler, to disassemble all section content and send the output to a file. This command is global and case sensitive. By default this checkbox is clear.

	Option	Description
2.	Disassemble Executable Section Content (-d)	Check to specify the **-d** command to the disassembler, to disassemble all executable content and send output to a file. By default this checkbox is checked.
3.	Intermix Source Code With Disassembly (-S)	Check to specify the **-S** command to the disassembler, to convert jbsr into jsr. By default this checkbox is checked.
4.	Display All Header Content (-x)	Check to specify the **-x** command to the disassembler, to display the contents of all headers. By default this checkbox is checked.
5.	Display Archive Header information (-a)	Check to specify the **-a** command to the disassembler, to display the archive header information. By default this checkbox is inactive.
6.	Display Overall File Header content (-f)	Check to specify the **-f** command to the disassembler, to display the contents of the overall file header. By default this checkbox is inactive.
7.	Display Object Format Specific File Header Contents (-p)	Check to specify the **-p** command to the disassembler, to display the file header contents and object format. By default this checkbox is inactive.
8.	Display Section Header Content (-h)	Check to specify the **-h** command to the disassembler, to display the section header of the file. By default this checkbox is inactive.
9.	Display Full Section Content (-s)	Check to specify the **-s** command to the disassembler, to display the full section of the file. By default this checkbox is clear.
10.	Display Debug Information (-g)	Check to specify the **-g** command to the disassembler, to display debug information in the object file. By default this checkbox is clear.
11.	Display Debug Information Using ctag Style (-e)	Check to specify the **-e** command to the disassembler, to display debug information using the ctags style. By default this checkbox is clear.
12.	Display STABS Information (-G)	Check to specify the **-G** command to the disassembler, to display any STABS information in the file, in raw form. By default this checkbox is clear.

	Option	Description
13.	Display DWARF Information (-W)	Check to specify the **-W** command to the disassembler, to display any DWARF information in the file. By default this checkbox is clear.
14.	Display Symbol Table Content (-t)	Check to specify the **-t** command to the disassembler, to display the contents of the symbol tables. By default this checkbox is clear.
15.	Display Dynamic Symbol Table Content (-T)	Check to specify the **-T** command to the disassembler, to display the contents of the dynamic symbol table. By default this checkbox is clear.
16.	Display Relocation Entries (-r)	Check to specify the **-r** command to the disassembler, to display the relocation entries in the file. By default this checkbox is clear.
17.	Display Dynamic Relocation Entries (-R)	Check to specify the **-R** command to the disassembler, to display the dynamic relocation entries in the file. By default this checkbox is clear.

SPT Assembler

SPT Assembler section is available for the following microcontroller projects:

- MPC577xK (SPT1)
- S32R274 (SPT2)
- S32R372 (SPT2.5).

SPT Assembler Panel

This section describes how to specify the SPT assembler behavior in the build properties for C/C++ project. The table below lists and describes the various options available on the Standard S32DS SPT assembler panel.

Table 33: Tool Settings - SPT Assembler pane

	Option	Description
1.	Command	Shows the location of the SPT assembler executable file. Default: **as-spt**
2.	All options	Shows the actual command line the SPT assembler will be called with. Default: empty
3.	Command line pattern	Shows the expert settings command line parameters. Default: **${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}**

General

This section describes how to specify the SPT assembler settings in the build properties for C/C++ project. The table below lists and describes the various options available on the General panel.

Table 34: Tool Settings - SPT Assembler - General

	Option	Description
1.	Assembler flags	Specify the flags that need to be passed with the SPT assemblerDefault: empty
2.	Include paths (-I)	This option changes the build target's search order of access paths to start with the system paths list.Default: empty
3.	Suppress warnings (-W)	This enables some extra warning flags that are not enabled by -Wall.By default this checkbox is clear.
4.	Announce version (-v)	Check this option if you want the S32DS Power v2017.R1 to show each command-line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit. This setting is equivalent to specifying the -v command-line option. The IDE displays just error messages that the compiler emits. The IDE suppresses warning and informational messages. By default this checkbox is unchecked.

SPT Disassembler

SPT Disassembler section is available for the following microcontroller projects:

- MPC577xK (SPT1)
- S32R274 (SPT2)
- S32R372 (SPT2.5).

SPT Disassembler Panel

This section describes how to specify the SPT disassembler behavior in the build properties for C/C++ project. The table below lists and describes the various options available on the SPT Disassembler panel.

Table 35: Tool Settings - SPT Disassembler pane

	Option	Description
1.	Command	Shows the location of the disassembler executable file. Default: **${cross_prefix}objdump${cross_suffix}**
2.	All options	Shows the actual command line the disassembler will be called with. Default: **-d -S -x**
3.	Command line pattern	Shows the expert settings command line parameters. Default: **${COMMAND} ${FLAGS} ${INPUTS}**

Settings

This section describes how to specify the SPT disassembler settings in the build properties for C/C++ project. The table below lists and describes the various options available on the Settings panel.

Table 36: Tool Settings - SPT Disassembler - Settings

	Option	Description
1.	Disassemble All Section Content (including debug information) (-D)	Check to specify the -D command to the SPT disassembler, to disassemble all section content and send the output to a file. This command is global and case sensitive. By default this checkbox is clear.
2.	Disassemble Executable Section Content (-d)	Check to specify the -d command to the SPT disassembler, to disassemble all executable content and send output to a file. By default this checkbox is checked.
3.	Intermix Source Code With Disassembly (-S)	Check to specify the -S command to the SPT disassembler, to convert jbsr into jsr. By default this checkbox is checked.
4.	Display All Header Content (-x)	Check to specify the -x command to the SPT disassembler, to display the contents of all headers. By default this checkbox is checked.
5.	Display Archive Header information (-a)	Check to specify the -a command to the SPT disassembler, to display the archive header information. By default this checkbox is inactive.
6.	Display Overall File Header content (-f)	Check to specify the -f command to the SPT disassembler, to display the contents of the overall file header. By default this checkbox is inactive.
7.	Display Object Format Specific File Header Contents (-p)	Check to specify the -p command to the SPT disassembler, to display the file header contents and object format. By default this checkbox is inactive.
8.	Display Section Header Content (-h)	Check to specify the -h command to the SPT disassembler, to display the section header of the file. By default this checkbox is inactive.
9.	Display Full Section Content (-s)	Check to specify the -s command to the disassembler, to display the full section of the file. By default this checkbox is clear.
10.	Display Debug Information (-g)	Check to specify the -g command to the SPT disassembler, to display debug information in the object file. By default this checkbox is clear.
11.	Display Debug Information Using ctag Style (-e)	Check to specify the -e command to the SPT disassembler, to display debug information using the ctags style. By default this checkbox is clear.

	Option	Description
12.	Display STABS Information (-G)	Check to specify the -G command to the SPT disassembler, to display any STABS information in the file, in raw form. By default this checkbox is clear.
13.	Display DWARF Information (-W)	Check to specify the -W command to the SPT disassembler, to display any DWARF information in the file. By default this checkbox is clear.
14.	Display Symbol Table Content (-t)	Check to specify the -t command to the SPT disassembler, to display the contents of the symbol tables. By default this checkbox is clear.
15.	Display Dynamic Symbol Table Content (-T)	Check to specify the -T command to the SPT disassembler, to display the contents of the dynamic symbol table. By default this checkbox is clear.
16.	Display Relocation Entries (-r)	Check to specify the -r command to the SPT disassembler, to display the relocation entries in the file. By default this checkbox is clear.
17.	Display Dynamic Relocation Entries (-R)	Check to specify the -R command to the SPT disassembler, to display the dynamic relocation entries in the file. By default this checkbox is clear.

SPT Preprocessor

SPT Preprocessor section is available for the following microcontroller projects:

- MPC577xK (SPT1)
- S32R274 (SPT2)
- S32R372 (SPT2.5).

SPT Preprocessor panel

This section describes how to specify the SPT preprocessor behavior in the build properties for C/C++ project. The table below lists and describes the various options available on the SPT Preprocessor panel.

Table 37: Tool Settings - SPT Preprocessor pane

	Option	Description
1.	Command	Shows the location of the preprocessor executable file. Default: **${cross_prefix}cpp${cross_suffix}**
2.	All options	Shows the actual command line the preprocessor will be called with. By default this field is empty
3.	Command line pattern	Shows the expert settings command line parameters. Default: **${COMMAND} ${FLAGS} ${INPUTS} ${OUTPUT_FLAG} ${OUTPUT}**

Includes

This section describes how to specify the directories paths and files paths in the build properties for Power Architecture project. The table below lists and describes the various options available on the **Includes** panel.

Table 38: Tool Settings - SPT Preprocessor - Includes

	Option	Description
1.	Include paths (-I)	This option adds the directory to the list of directories to be searched for header files. Directories named by -I are searched before the standard system include directories. If the directory is a standard system include directory, the option is ignored to ensure that the default search order for system directories and the special treatment of system headers are not defeated. This option changes the build target's search order of access paths to start with the system paths list. The preprocessor can search **#include** files in several different ways. You can also set the search order as follows: For include statements of the form **#include"xyz"**, the preprocessor first searches user paths, then the system paths. For include statements of the form **#include<xyz>**, the preprocessor searches only system paths. This option is global. By default this field is empty
2.	Include files (-include)	Use this option to specify the include file search path. Process file as **if #include "file"** appeared as the first line of the primary source file. However, the first directory searched for file is the preprocessor's working directory instead of the directory containing the main source file. If not found there, it is searched for in the remainder of the **#include "..."** search chain as normal. If multiple **-include** options are given, the files are included in the order they appear on the command line. By default this field is empty

Symbols

This section describes how to specify the preprocessor behavior in the build properties for Power Architecture project. The table below lists and describes the various options available on the **Symbols** panel.

Table 39: Tool Settings - SPT Preprocessor - Symbols

	Option	Description
1.	Defined symbols (-D)	Use to specify the substitution strings that the preprocessor applies to all the assembly-language modules in the build target. Enter just the string portion of a substitution string. The S32DS Power v2017.R1 prepends the **-D** token to each string that you enter. For example, entering **opt1 x** produces this result on the command line: **-Dopt1 x** **Note:** This option is similar to the **#define** directive, but applies to all assembly-language modules in a build target By default this field is empty
2.	Undefined symbols (-U)	Cancel any previous definition of name, either built in or provided with a **-D** option By default this field is empty

Toolchain customization

User can specify which tools the builder needs to include when it builds the project for a specified toolchain and configuration. Normally, users need not to edit toolchains manually.

To customize the toolchain used in your build configuration, perform these steps:

1. Follow the steps listed in Section.
2. Select from build properties list the **Toolchain Editor** item. The **Toolchain Editor** pane appears on the right. The **Used tools** list presents the set of default tools, which are available by default just after the project creation.
3. Define toolchain settings. The settings are listed in the C/C++ Development User Guide (see **Help** > **Help Contents** > **Third Party References** > **C/C++ Development User Guide** > **Reference** > **C/C++ Properties** > **C/C++ File properties** > **C/C++ Build** > **Tool chain editor page**.

   Note: Tools from the set generated after the project creation cannot be removed from this set. Tools which were added manually to toolchain can be deleted by using the **Select Tools** window.

Toolchains from different vendors

The **GHS PowerPC Standalone Executable Toolchain** (GHS) and **Wind River Diab** (Diab) toolchains are available if corresponding Eclipse plug-ins was installed. To install these plug-ins use the **Install new software** feature.

S32DS Power v2017.R1 and the **New S32DS Project** wizard provide support only for the latest available versions of the toolchains:

- GHS PowerPC Standalone Executable Toolchain 2015_14+
- Wind River Diab 5.9.4.7.

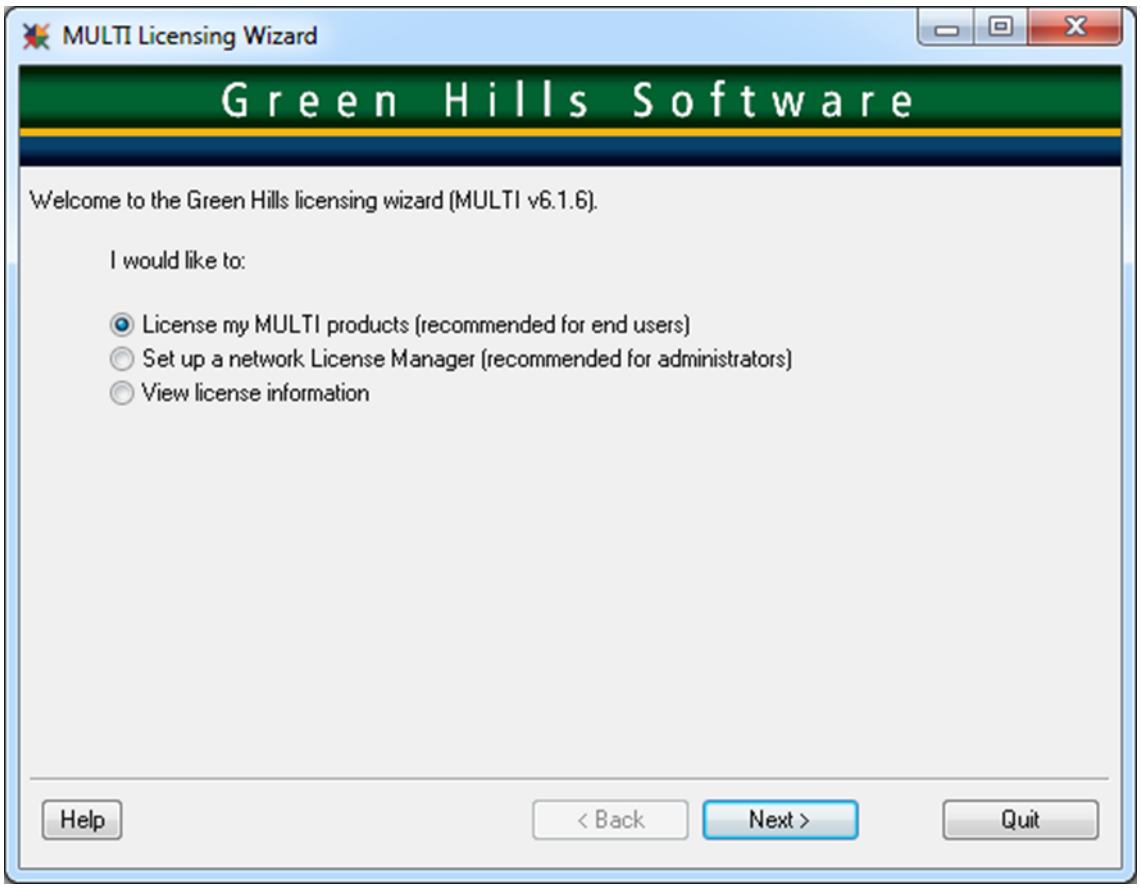
The user needs to have the relevant toolchain properly installed and licensed on the computer.

GHS toolchain

GHS plug-in installation

To install GHS plug-in:

1. Install the MULTI IDE software (You need to register on GHS website support.ghs.com and purchase MULTI IDE software).
2. Request online license via MULTI licensing wizard which will be launched automatically at first launch of MULTI IDE:



GHS support will send license file by e-mail (usually in 1-2 days) with license installation details.

The plug-in comes with GHS installer and is located after installation in corresponding GHS directory, for example C:\ghs_ppc\comp_201514\eclipse.

3. Use the S32DS Power v2017.R1 **Install new software** feature to install GHS plug-in.
4. Select the **GreenHillsMULTI for Eclipse (X86)** item.

   GHS plug-in uses CDT internal builder by default. For the same user experience the **New S32DS Project** wizard sets make builder as current builder in **Project Properties** > **C/C++** > **Build** > **Tool Chain Editor** and add corresponding path in PATH variable.

   For more information, see GHS website: support.ghs.com.

Bareboard project settings

This section describes how to specify the GHS compiler behavior in the build properties for Power Architecture project.

Table 40: Bareboard project settings - GHS compiler

	Option	Description
Target	-cpu=ppc5777mz425	Set CPU (see table below)
Assembler	-g	Generate debug information
	-dwarf2	Set DWARF-2 format
	-preprocess_assembly_files	Preprocess assembler files
Compiler	-g	Generate debug information

	Option	Description
	-dwarf2	Set DWARF-2 format
	-Idirectory	Include directories
	-Ldirectory	Library directories
Linker	-T../Project_Settings/Linker_Files/memory.ld	Linker script file
	-archive –o name	Generate library

Linker files must be provided while it offers to change memory allocation for multicore targets. Linker files syntax a little bit different from GCC syntax (e.g. ALIGN option). Startup code syntax is similar to GCC except comments, section and alignment options.

CPU options

This section describes CPU options for target MCU.

Table 41: CPU options for target MCU

	Z0 cores	Z2 cores	Z4 cores	Z7 cores
MPC5601D/2D	ppc560xd			
MPC5602B/3B/4B/5B/6B/7B	ppc560xb			
MPC5602C/3C/4C	ppc560xc			
MPC5601P/2P/3P/4P	ppc560xp			
MPC5606S	ppc560xs			
MPC5644B/5B/6B			ppc564xb	
MPC5644C/5C/6C	ppc564xcz0		ppc564xc	
MPC5741P/2P/3P/4P			ppc5744p ppc5744pz425	
MPC5743R			ppc5746r	
MPC5745C/6C		ppc574xcz210	ppc574xcz4204	
MPC5745B/6B			ppc574xb	
MPC5745D			ppc574xb	
MPC5745R			ppc5746r	
MPC5746R			ppc5746r	
MPC5746D		ppc574xcz210	ppc574xcz4204	
MPC5746G/7G		ppc574xgz210	ppc574xgz4204	
MPC5748G		ppc5748gz210	ppc5748gz4204	
MPC5774K/75K			ppc577xkz4201	ppc577xkz7260
MPC5777C				ppc5777c
MPC5777M		ppc5777mz425		ppc5777mz710 ppc5777mz720

Diab toolchain

Diab plug-in installation

To install Diab plug-in:

1. Register on Wind River website www.windriver.com and purchase Wind River Diab Compiler software.
2. Start installation of the Wind River Diab Compiler software.
3. Request Wind River Diab Compiler license: When installing the Wind River Diab Compiler use the **Permanent activation** option during the **Choose your activation type for this installation** step. Browse for the **install.txt** file which contains all needed installation keys. Continue with the **Next** step.

4. Select **Wind River Compiler, Architecture: PowerPC** and **Wind River Diab Eclipse CDT Plug-in**.

   The Windriver installer comes with the plug-in. The plug-in (**Diab Compiler Support for Eclipse C/C++ IDE**) is located after installation in corresponding folder, by default **C:\WindRiver\diab\eclipse_cdt**.
5. Use **Install** > **Available Software** to install the plug-in.

   After installation plug-in automatically detects installed versions of Diab.

   See the **Plugin for Eclipse CDT getting started** document from installation folder for detailed plug-in installation instruction.

   Plugin uses CDT internal builder by default. For the same user experience the **New S32DS Project** wizard sets make builder as current builder in **Project Properties** > **C/C++ Build** > **Tool Chain Editor** and add corresponding path in PATH variable.

   For more information, see Wind River website: www.windriver.com/products/development-tools/#diab_compiler.

Bareboard project settings

This section describes how to specify the Diab compiler behavior in the build properties for MPC5777M Z4 and Z7 cores.

Table 42: Bareboard project settings - Diab compiler

	Option	Description
Target	-tPPCE200Z4VEF:simple -tPPCE200Z7VEF:simple	Set CPU (see table below)
Assembler	-Xpreprocess-assembly	Preprocess assembler files
	-g	Generate debug information
	-Xdebug-dwarf2	Set DWARF-2 format
	-Xisa-vle	Use VLE instruction set
Compiler	-Xdebug-dwarf2	Set DWARF-2 format
	-Xdebug-local-cie	Generate a local Common Information Entry (CIE) for each unit. The option must be set for correct work with GDB. Otherwise GDB client will fault during debug session start
	-Idirectory	Include directories
	-Ldirectory	Library directories
Linker	../Project_Settings/ Linker_Files/ default.ld	Linker script file

Linker files must be provided while it offers to change memory allocation for multicore targets. Startup code syntax is similar to GCC. Diab doesn't have an option to create library instead of ELF.

CPU options

This section describes CPU options for target MCU.

Table 43: CPU options for target MCU

	Z0 cores	Z2 cores	Z4 cores	Z7 cores
MPC5601D/2D	PPCE200Z0HV			
MPC5602B/	PPCE200Z0V			
3B/4B/5B/6B/7B				
MPC5602C/3C/4C	PPCE200Z0V			
MPC5601P/ 2P/3P/4P	PPCE200Z0V			
MPC5606S	PPCE200Z0HV			
MPC5644B/5B/6B			PPCE200Z4DV	
MPC5644C/5C/6C	PPCE200Z0HV		PPCE200Z4DV	
MPC5741P/2P/3P/4P			PPCE200Z4201N3V	

	Z0 cores	Z2 cores	Z4 cores	Z7 cores
MPC5743R			PPCE200Z425N3V	
MPC5745C/6C		PPCE200Z210N3V	PPCE200Z420N3V	
MPC5745B/6B			PPCE200Z420N3V	
MPC5745D			PPCE200Z420N3V	
MPC5745R			PPCE200Z425N3V	
MPC5746R			PPCE200Z425N3V	
MPC5746D		PPCE200Z210N3V	PPCE200Z420N3V	
MPC5746G/7G		PPCE200Z210N3V	PPCE200Z4204N3V	
MPC5748G		PPCE200Z210N3V	PPCE200Z4204N3V	
MPC5774K/75K			PPCE200Z4201N3V	PPCE200Z7260N3V
MPC5777C				PPCE200Z759N3V
MPC5777M			PPCE200Z425BN3V	PPCE200Z710N3V PPCE200Z720N3V

View/manage resources in build configurations

Resources can be viewed/ managed in the **Build configurations** dialog. For open **Build configurations** dialog:

1. Select the project in the **Project Explorer** view.
2. Call the context menu by right click.
3. Select **Build Configurations Explorer** in the context menu. The **Build configurations** dialog will be displayed.

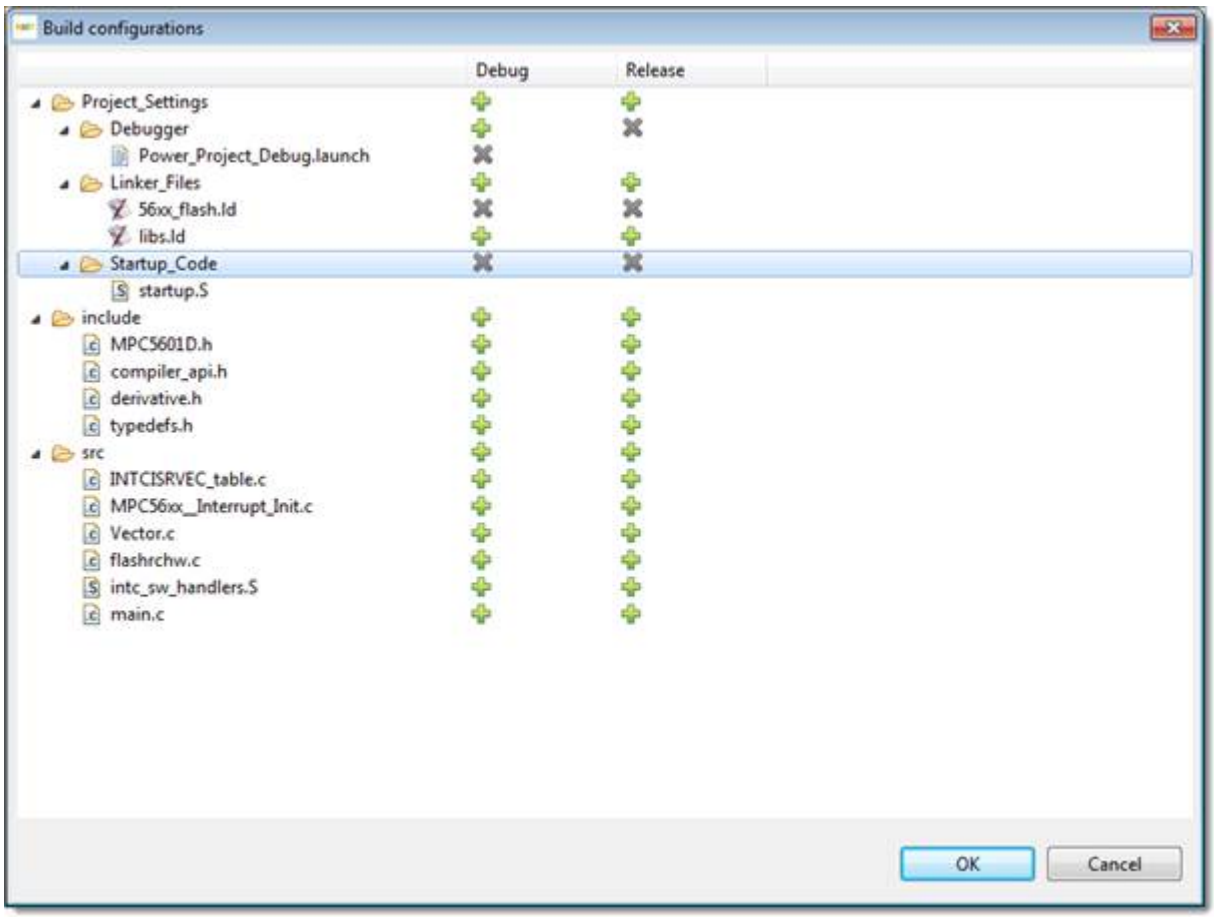
   A tree of project elements is displayed on the left part of the window. Folders and files compose a hierarchy of the project elements. Build configurations can be viewed or modified by selecting files.

   The columns corresponding to build configuration are presented on the right from the elements tree. In our example we have two configurations: **Debug** and **Release**.

   Green plus or grey cross signs can be set in each column:

   - Green plus sign - the checked element is included in the build configuration (configuration name is the column header);
   - Grey cross sign - the checked element is excluded from the configuration.

   To change the state of the selected element, click on the icon in configuration column.


Build configurations
Debug
Release
Project_Settings
Debugger
Power_Project_Debug.launch
Linker_Files
56xx_flash.ld
libs.ld
Startup_Code
startup.S
include
MPC5601D.h
compiler_api.h
derivative.h
typedefs.h
src
INTCISRVEC_table.c
MPC56xx_Interrupt_Init.c
Vector.c
flashrchw.c
intc_sw_handlers.S
main.c
OK
Cancel

Chapter 4

Working with debugger

Topics:

S32DS Power v2017.R1 project can have multiple associated launch configurations. A launch configuration is a named collection of settings that the S32DS Power v2017.R1 tools use.

Customizing launch configuration

The **Debug Configurations** dialog box contains several tabs allowing you to customize launch configurations.

Note: As you modify a debug configuration's settings, you create pending, or unsaved, changes to that debug configuration. To save the pending changes, you must click the **Apply** button at the bottom of the **Debug Configurations** dialog box, or click the **Close** button and then the **Yes** button of the **Save Changes** dialog box.

Note: You can revert pending changes and restore their last saved settings. To undo pending changes, click the **Revert** button at the bottom of the **Debug Configurations** dialog box. The S32DS Power v2017.R1 restores the last set of saved settings to all pages of the **Debug Configurations** dialog box. Also, the IDE disables the **Revert** button until you make new pending changes.

The settings of debug configuration depend on the used connection type (GDB Hardware or GDB PEMicro Interface)

How to customize debug configuration for GDB PEMicro Interface see P&E GDB Server Plug-In for e200 Devices. Debug Configuration User Guide in the S32DS Power v2017.R1 Help (**Freescale S32 Design Studio for Power Architecture, Version 2017.R1** > **Common Manuals**).

Main

Use the **Main** pane to specify the project and the application you want to run or debug. The **Main** tab presents groups of options for specifying different settings.

Table 44: Main tab options

Group	Option	Description
N/A These options are ungrouped.	**Project**	Specifies the project to associate with the selected debug launch configuration. Click **Browse** to select a different project.
	Specify the number of additional Elf Files you wish to program	Enter the number of additional Elf-files and click **Generate Elf Fields**. Specify path to the additional elf(-s) in the field(-s) **Specify Additional Elf <elf number>.**
	C/C++ application	Specifies the name of the C or C++ application
	Variables…	Click to open the **Select Variable** dialog box and choose the build variables to be associated with the program. See details in **Common Features Guide** (**Environment variables in debug configuration** topic)
	Search Project…	Click to open the Program Selection dialog box and select a binary
Build (if required) before launching		Controls how auto build is configured for the launch configuration. Changing this setting overrides the global workspace setting and can provide some speed improvements.
	Build configuration	Specify the build configuration either explicitly, or use the current active configuration. By default, the list is set to the configuration selected in the tree of debug configurations available in the left pane of **Debug Configurations** window.
	Enable auto build	Always build project before launching. This may slow down launch performance. By default, this check box is cleared.

Group	Option	Description
	Disable auto build	Disables auto build for the debug configuration. No build action will be performed before starting the debug session. Requires manually building project before launching. This may improve launch performance. By default, this check box is cleared.
	Use workspace settings	Uses the global auto build settings. By default this check box is selected.
	Configure Workspace Settings…	The hyperlink opens the **Launching** preference panel where you can change the workspace settings. It will affect all projects that do not have project specific settings.

Debugger

Use the **Debugger** pane to tune a debugger to use when debugging an application. The **Debugger** tab presents groups of options for specifying different settings.

Note:

The options set under the Debugger tab change depending on the derivative and connection selected while creating the project.

About connections settings see Connections chapter.

If the user creates the new debug configuration (for existed project or after elf import) then following settings shall be checked for *GDB PnE Micro Interface* debug configuration - the device and GDB client are set correctly, so the **Target** field and **Executable** text boxes the **Debugger** pane should not be empty.

Startup

Use the **Startup** pane to specify specific options used to configure the debug session.

Note: The options set under the **Startup** tab change depending on the derivative and connection selected while creating the project.

Source

Use the **Source** pane to specify the location of source files used when debugging a C or C++ application. By default, this information is taken from the build path of your project. The **Source** tab options are explained in the following table.

Note: The options set under the **Source** tab change depending on the derivative and connection selected while creating the project.

Common

Use the **Common** pane to specify the location to store your run configuration, standard input and output, and background launch options. The **Common** tab presents groups of options for specifying different settings.

Note: The options set under the **Common** tab change depending on the derivative and connection selected while creating the project.

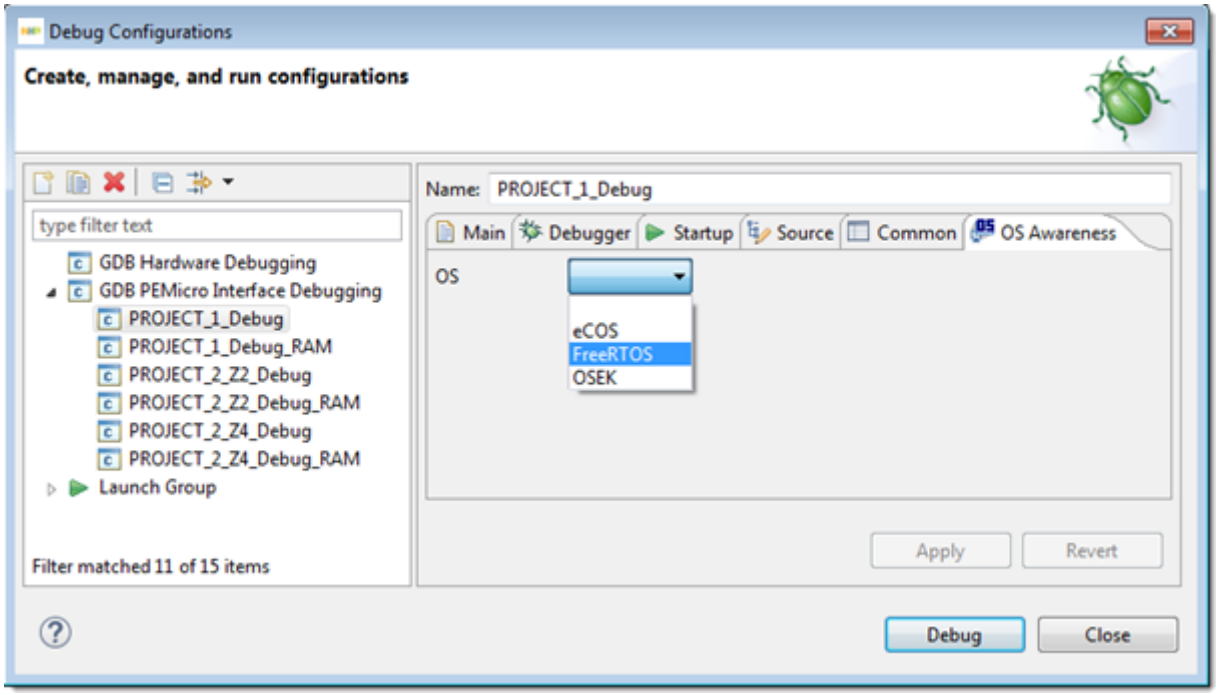
OS Awareness

The **OS Awareness** tab enables you to inform the debugger of the operating system (OS) the target is running. This enables the debugger to provide additional functionality specific to the selected OS.

Use the **OS Awareness** pane to select the **OS** option. The **OS Awareness** tab presents one drop-down list. The next options are available in the drop-down list:

- FreeRTOS

- OSEK.



OS awareness depends on having debug symbols for the OS loaded within the debugger.

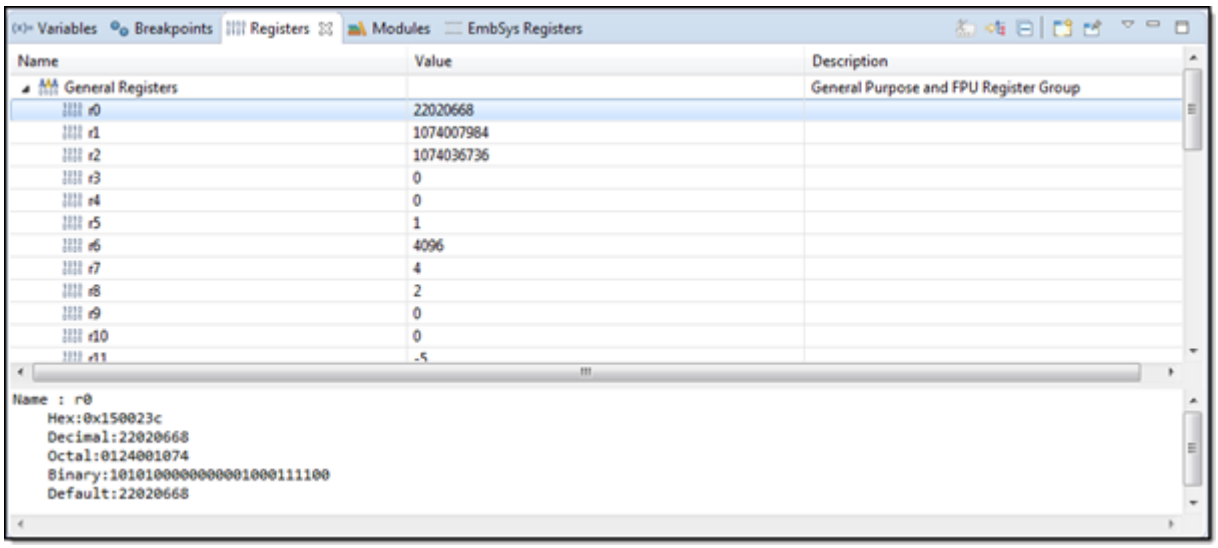
After selecting an **OS** the **OS Resources** view is available. See Common Features Guide for S32 Design Studio for Power Architecture, Version 2017.R1.

Debugging Bareboard Software

S32DS Power v2017.R1 cannot export/import register data to/from file and show registers information offline (without a debug session).

Displaying register contents

Use the **Registers** view to display and modify the contents of the registers of the processor on your target board. To display this view from the **Debug** perspective, select from the S32DS Power v2017.R1 menu bar**Window** > **Show View** > **Registers**, and the **Registers** view appears. The following figure shows the **Registers** view with the **General Registers** tree element expanded:



The **Registers** view displays categories of registers in a tree format. To display the contents of a particular category of registers, expand the tree element of the register category of interest.

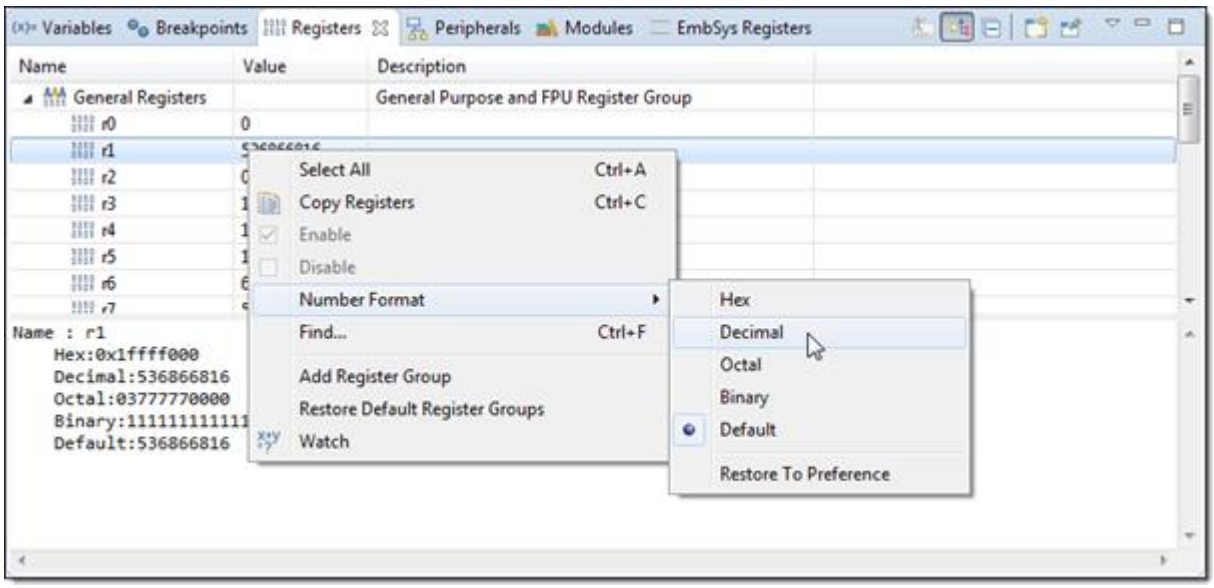
Changing register data display format

User can change the format in which the debugger displays the contents of registers. For example, user can specify that a register's contents be displayed in hexadecimal, rather than binary. The debugger provides these data formats:

- Default
- Decimal
- Hexadecimal
- Octal
- Binary

To change register display format:

1. Open the **Registers** view. Expand the hierarchical list to reveal the register for which you want to change the display format.
2. Select the register value that you want to view in a different format. The value highlights.
3. Right-click to display the pop-up menu and choose **Number Format** > **<data format>** from the context menu that appears, where **<data format>** is the data format in which you want to view the register value.



The selected register value changes format.

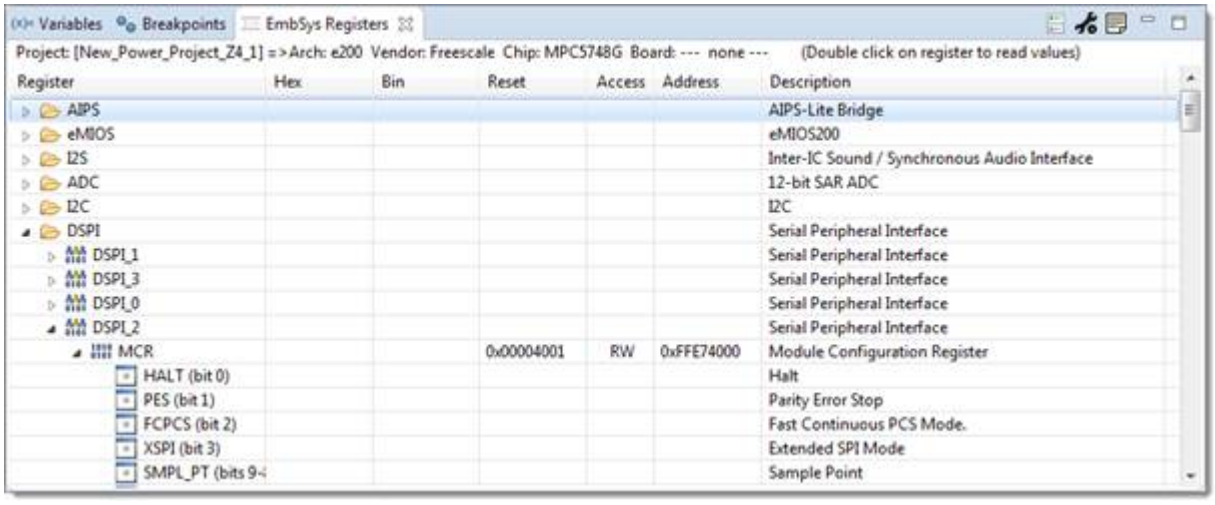
View peripheral registers

EMBedded **SYS**tems Registers view is designed for monitoring and modifying memory values of embedded devices. Therefore it offers a structured display of the special functions registers (SFR).

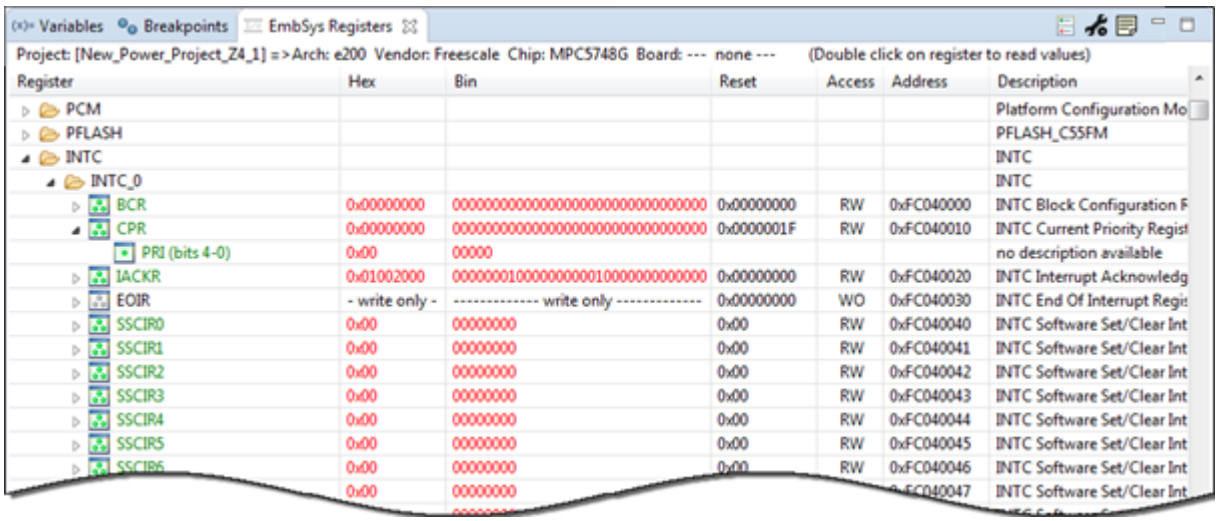
EmbSys Registers view is available on the **Debug** Perspective. To display this view from the **Debug** perspective, select **Window** > **Show View** > **Other** > **Debug** > **EmbSysRegisters** from the S32DS Power v2017.R1 menu bar, and the **EmbSys Registers** view appears.

The **EmbSys Registers** view displays categories of registers in a tree format. To display the contents of a particular category of registers, expand the tree element of the register category of interest.

The register values are presented in the Hexadecimal (**HEX**) and Binary (**Bin**) column of the view. As well as information about value out of Reset, Access type, Address and description:



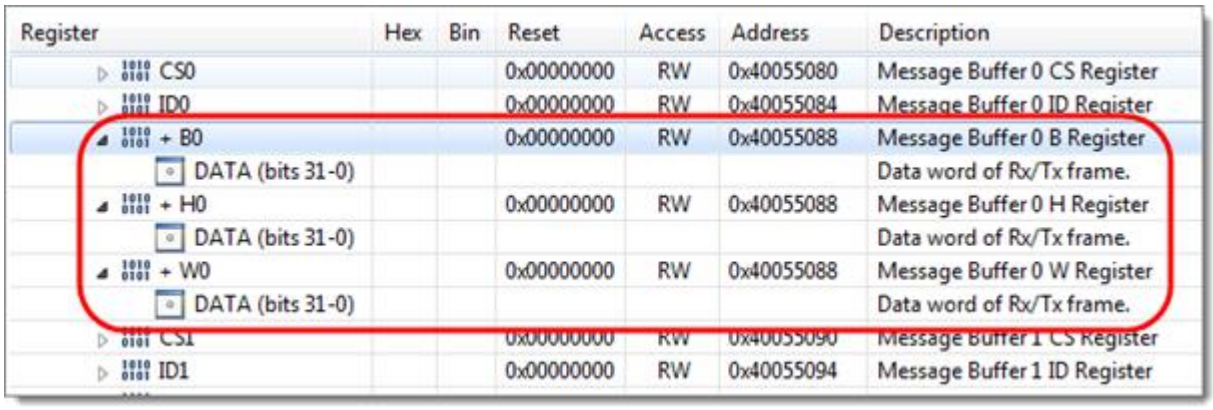
Bit level details as well available for registers. To read actual register value from target – double-click on the register name, the name of register will change its color to green, only one register will be read from memory on each debug action like step, resume/stop, breakpoint. If value of register changed, its color will change to red.



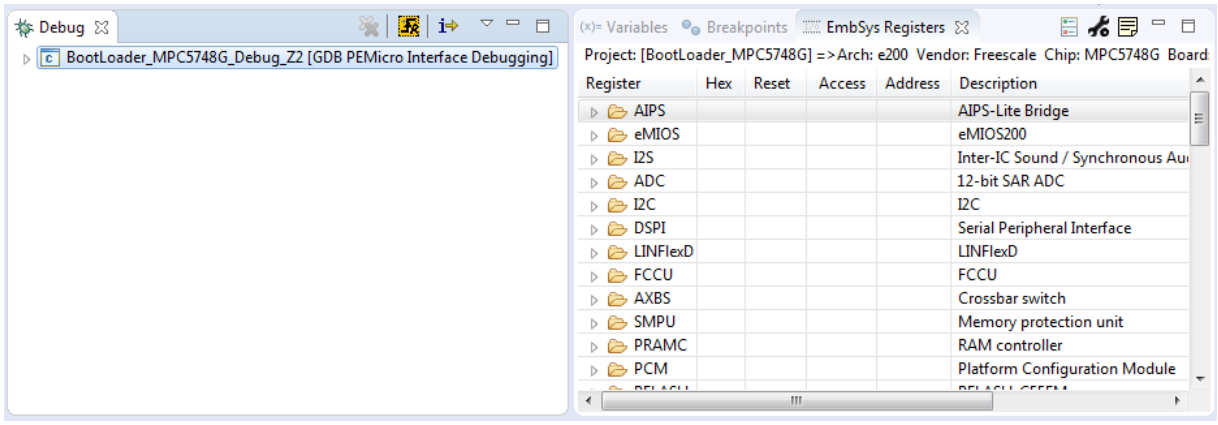
To stop reading the register value – double-click on its name again. To read entire group of registers – double-click on the group name.

Note: Be careful when selecting register group with big number of registers for reading as it might slow down the debugging.

The **Register** field can contain the "+" sign placed before the register name. These marked rows are aliases of a register having one common address. See example:

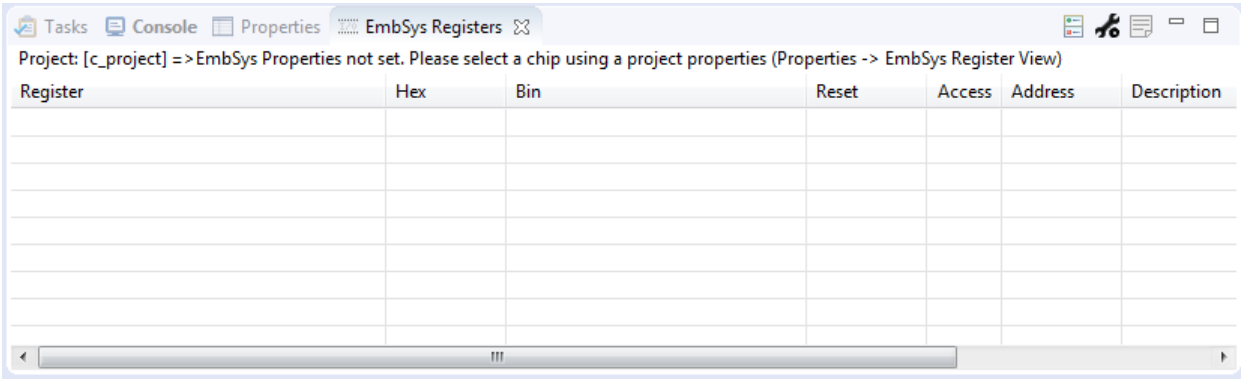


View is updated when user choose a project in **Project Explorer**. View displays register info including when debug session is off. View is updated by choosing debug context in the **Debug** view.



If different projects were chosen in some project's view and in the **Debug** view, **EmbSys Register** view is updated with the values of project, chosen in the just activated view.

If project, chosen in some project's view, have no settings for **EmbSys Register** view, view becomes empty and the instruction message is shown to user.



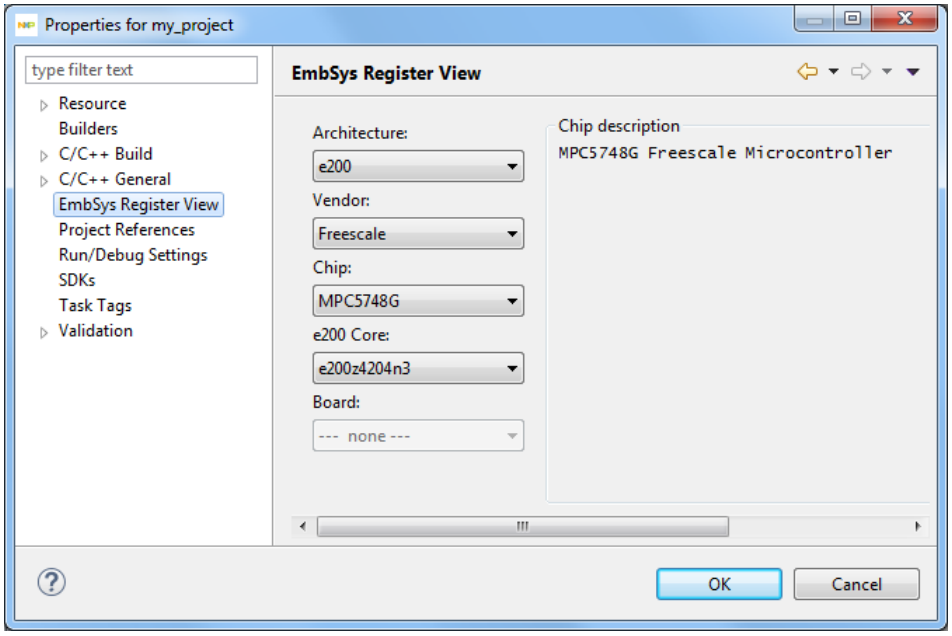
View responds to changes of project properties including when debug session is on.

The table below describes configuration buttons on the view toolbar.

Table 45: EmbSys Registers pane – toolbar buttons

Button	Description
Picked cherries	Filters and displays the selected rows of registers
EmbSysRegView Project Properties	Allows to directly access to the Properties for <project name> dialog box with XML-settings of a selected project
Copy selection to clipboard	Copies information from selected row (or rows) to clipboard: **Register**, **Hex**, and **Address** fields

XML data can be configured on project properties page.



View behavior can be configured for all projects in the global settings:

Window > **Preferences** > **C/C++** > **Debug** > **EmbSys Register View Behavior**:

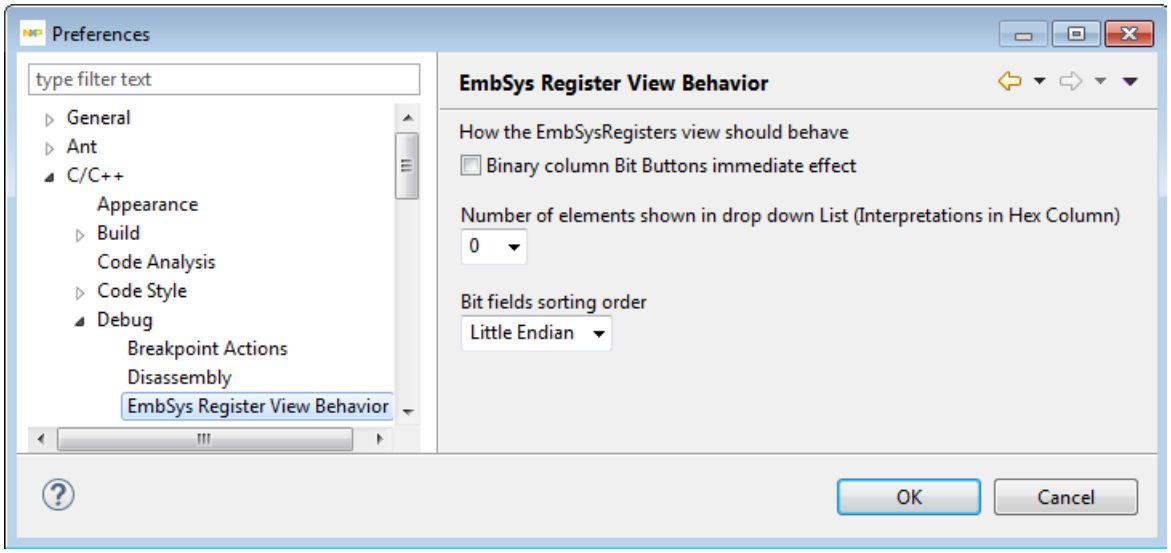


Table 46: Tool Settings - EmbSys Register View Behavior pane

	Option	Description
1.	Binary column Bit Buttons immediate effect	Check to enable Binary column Bit Buttons immediate effect
2.	Number of elements shown in the drop list (interpretation in the Hex column)	Use to specify number of elements shown in the drop list

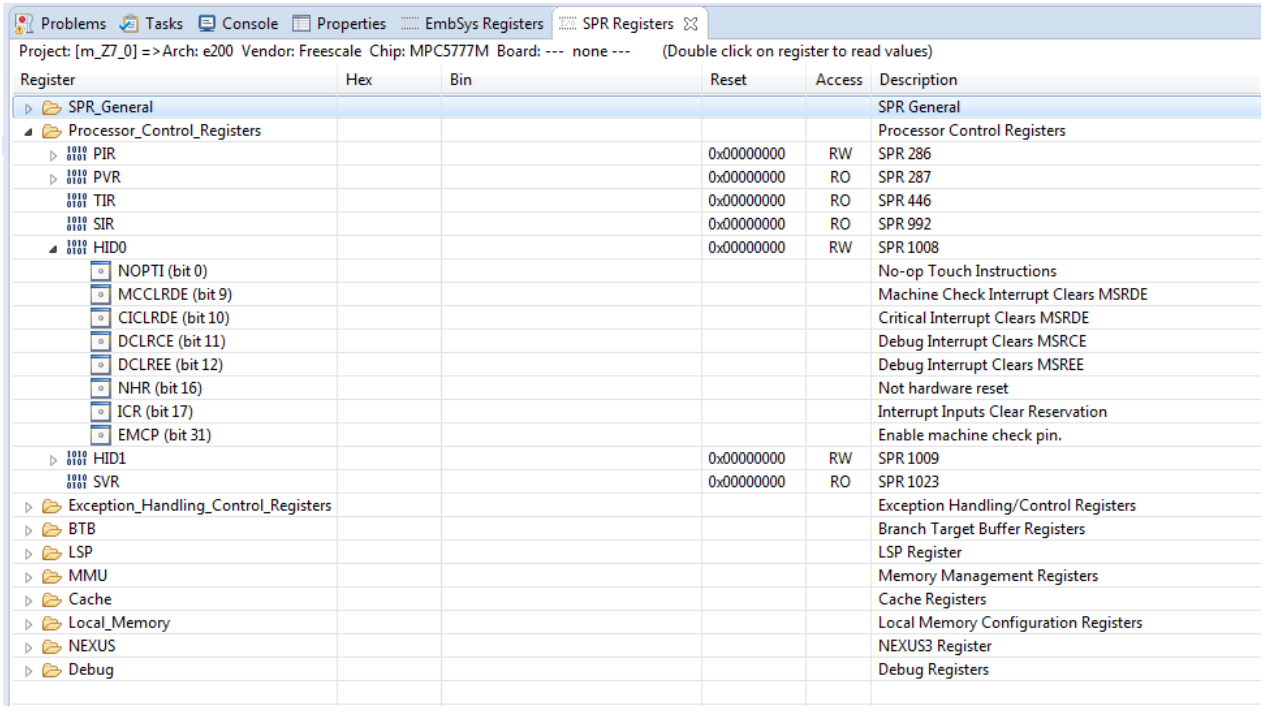
View SPR registers

The SPR Registers view is designed for monitoring and modifying memory values of embedded devices. Therefore it offers a structured display of the special purpose registers (SPR).

To display this view, select **Window** > **Show View** > **Other** > **Debug** > **SPR Registers** from the S32DS Power v2017.R1 menu bar, and the SPR Registers view appears.

The **SPR Registers** view displays categories of registers in a tree format. To display the contents of a particular category of registers, expand the tree element of the register category of interest.

The register values are presented in the Hexadecimal (**HEX**) and Binary (**Bin**) column of the view. As well as information about value out of Reset, Access type and description:



The functionality of **SPR Registers** view is the same as **EmbSys Registers** view (see View peripheral registers topic).

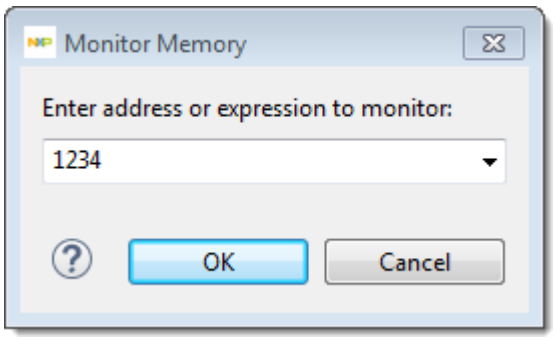
Viewing memory

Use the **Memory** view to examine the active memory rendering of a specified expression or address. The **Memory** view supports the display of multiple memory spaces. See details in **S32DS Power v2017.R1 Common Features Guide** (**Memory view** topic)

Adding memory monitor

You can add multiple memory monitors to the **Memory** view. To add a new memory monitor, perform these steps:

1. Start a debugging session.
2. Open the **Memory** view.
3. Click the plus-sign icon on the **Monitors** pane toolbar: Alternatively, right-click in the **Monitors** pane and select **Add Memory Monitor** from the context menu. The **Monitor Memory** dialog box appears.



4. Enter address or expression to monitor in decimal or hexadecimal values. You can use the drop-down list to select a previously specified expression.
5. Click **OK**. New memory monitor appears in the **Memory** view.

Adding memory rendering

You can use the Renderings pane of the **Memory** view to examine the memory content, starting at any valid address. The information displayed in this page is read only and cannot be used to modify the memory content.

To add a new memory rendering, perform these steps.

1. Start a debugging session.
2. Open the **Memory** view.
3. In the Monitors pane, select the memory monitor for which you want to add a memory rendering.
4. Click the **New Renderings…** tab to select renderings.
5. Select a rendering type from the **Select rendering(s) to create** list and click the **Add Rendering(s)** button. Alternatively, right-click in the Renderings pane and select **Add Rendering** from the context menu. The selected memory rendering type appears in the **Memory** view.

Removing memory rendering

To remove a memory rendering from the **Memory** view, perform these steps:

1. Open the **Memory** view.
2. In the Renderings pane, select the tab that corresponds to the memory rendering that you want to remove.
3. Click the cross-sign icon on the Renderings pane toolbar: ✖ . Alternatively, right-click on the Renderings pane and select **Remove Rendering** from the context menu. The memory rendering is removed from the **Memory** view.

Resetting to base address

To reset the memory rendering and display the base address of the rendering, perform these steps.

1. Open the **Memory** view.
2. In the Renderings pane, select the tab that corresponds to the disassembly rendering that you want to reset to the base address.
3. Right-click in the Renderings pane and select **Reset to Base Address** from the context menu.
4. The disassembly rendering scrolls to the line that contains the base address of the displayed rendering.

Go to address

The **Memory** view provides graphical controls to display memory at a specific address.

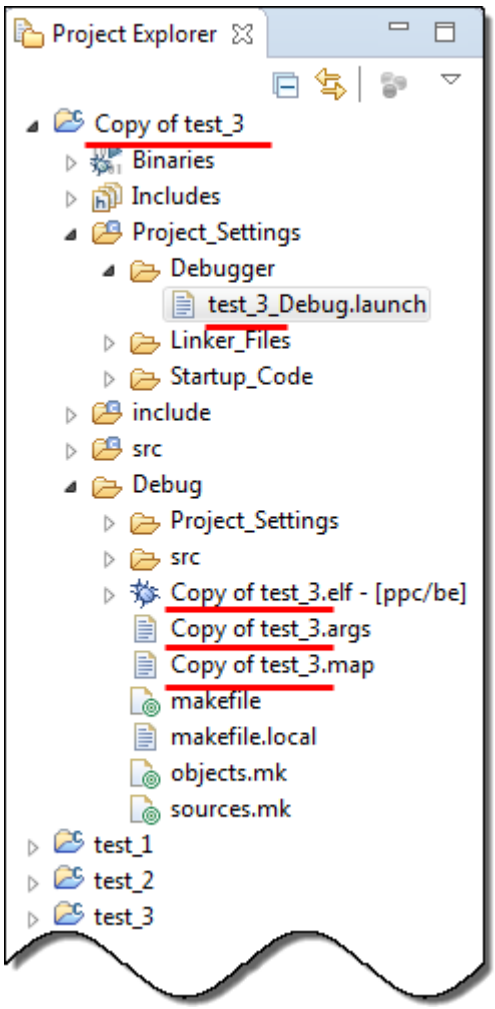
To go to a specific address, perform these steps:

1. Open the **Memory** view.
2. In the Renderings pane, select the tab that corresponds to the disassembly rendering for which you want to display a specific address.
3. Right-click in the Renderings pane and select **Go to Address…** from the context menu. A group of controls appears on the Renderings pane.
4. In the blank text box, enter the address that you want to display.

   Note: Check the **Input as Hex** checkbox only if you enter the address in hexadecimal notation.
5. Click **OK** to have the Disassembly rendering scroll to the specified address. Alternatively, click **Cancel** to abort the operation and hide the group of controls.

Prepare a copied project for debugging

For example, "**test_3**" project was copied to "**Copy of test_3**":

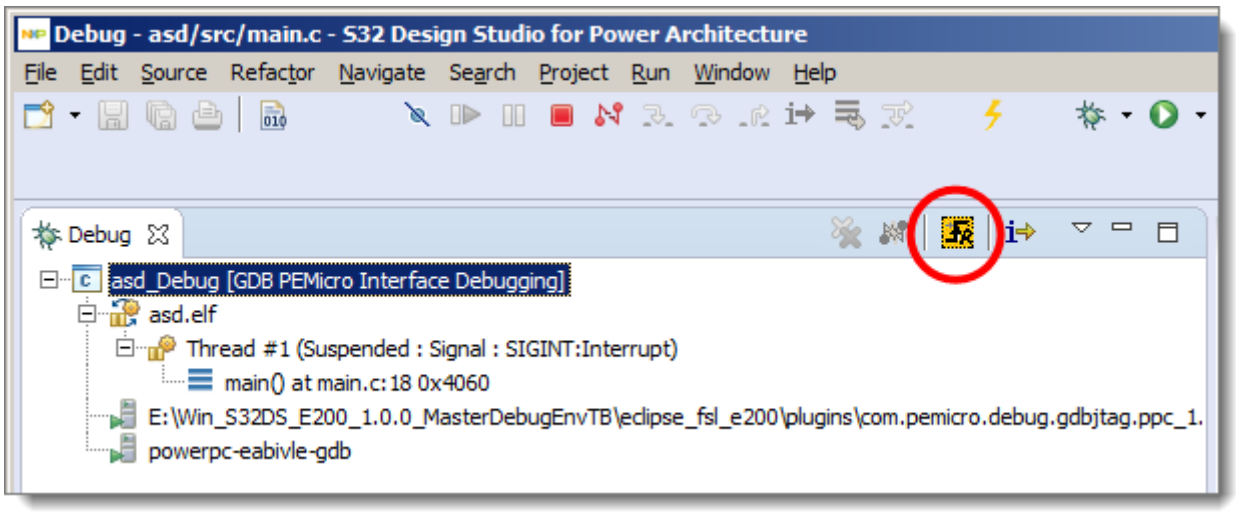


If you copy an project by selecting **Copy** and the **Paste** in the context menu, S32DS Power v2017.R1 automatically adds a numeric postfix to the name of the copied project: **Copy of test_4**.

Now the launch configuration is ready and project can be debugged.

Target resetting

Use the **Send HW Reset to target** button to reset your target board. To send HW Reset from the **Debug** perspective, select from the **Debug** toolbar the **Send HW Reset to target** button and control command execution in the **Console** view. The following figure shows the **Debug** view with the **Send HW Reset to target** button:



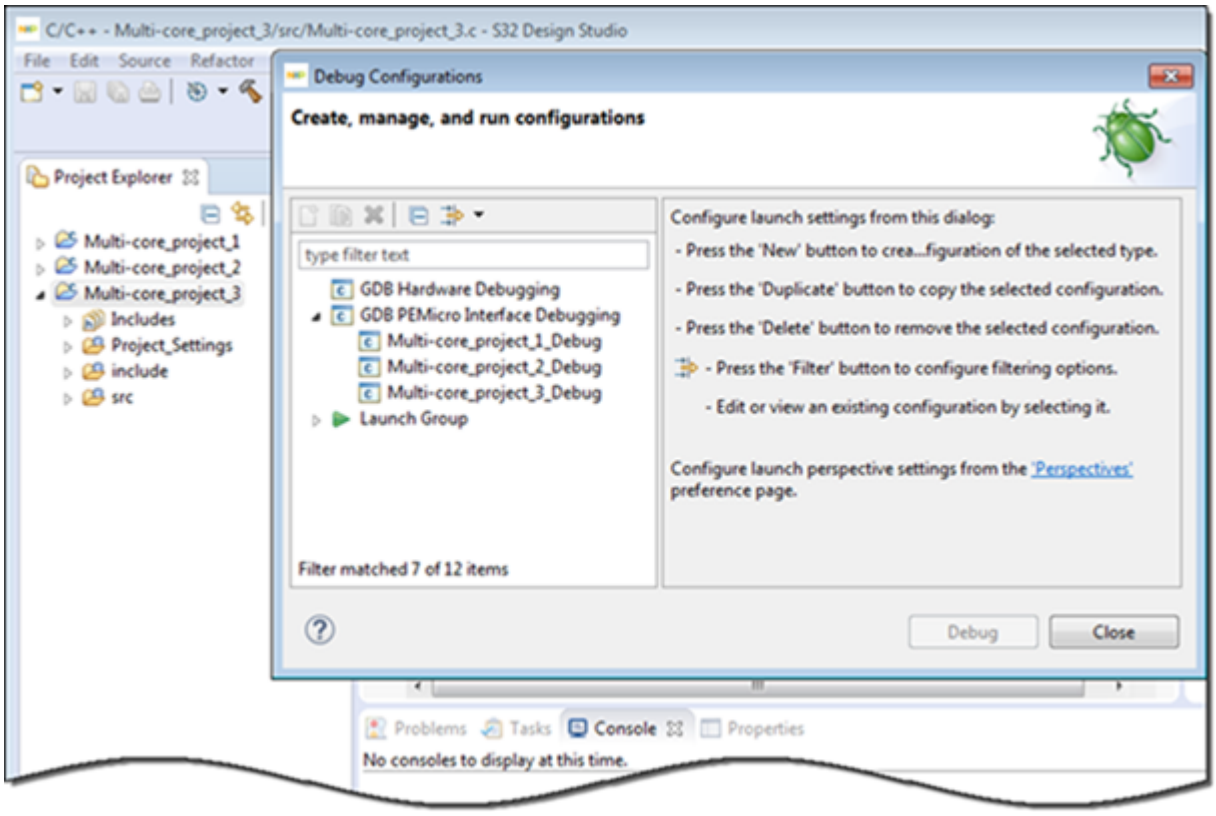
The following S32DS Power v2017.R1 devices do not support the Reset function:

- MPC5643L
- MPC567xK
- MPC564xB.

These devices rely on SSCM to turn on the core and do not have the ability to turn off or reset using a reset vector or mode entry method vector.

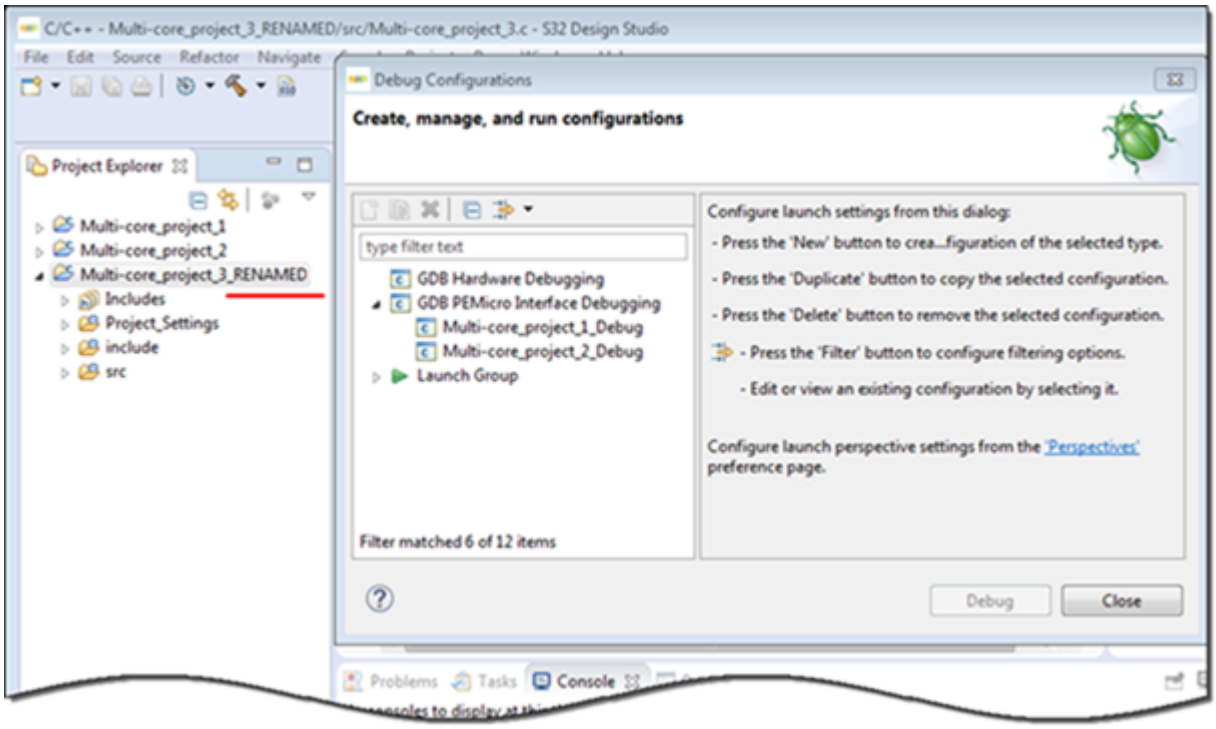
Prepare a renamed project for debugging

If you open the Debug Configuration window of the project there are configurations for the project:



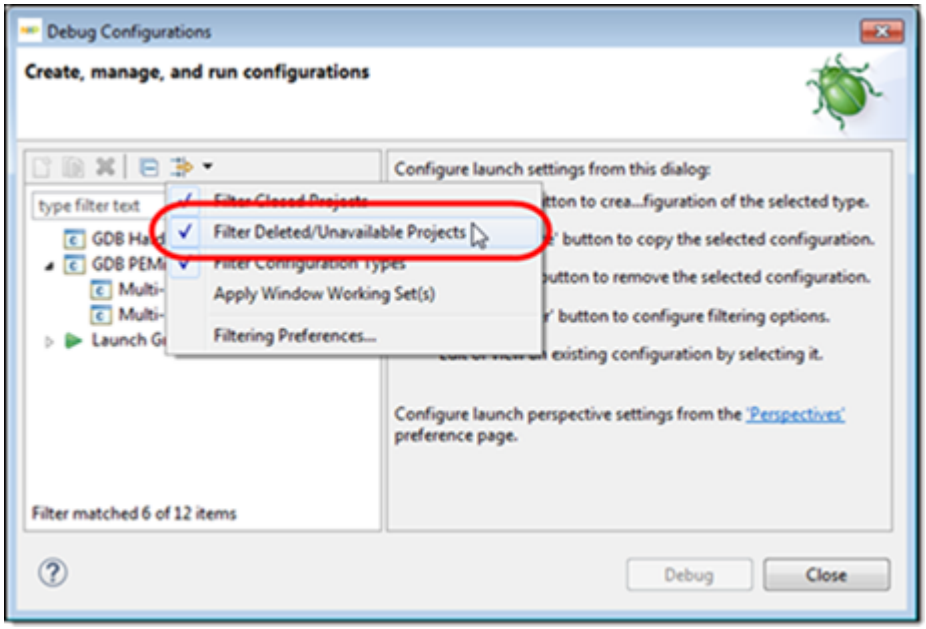
If you rename this project and open **Debug Configuration** window then there are no configurations for renamed project.

Example: If you rename the “Multi-core_project_3” project then there is no “Multi-core_project_3_Debug” configuration:



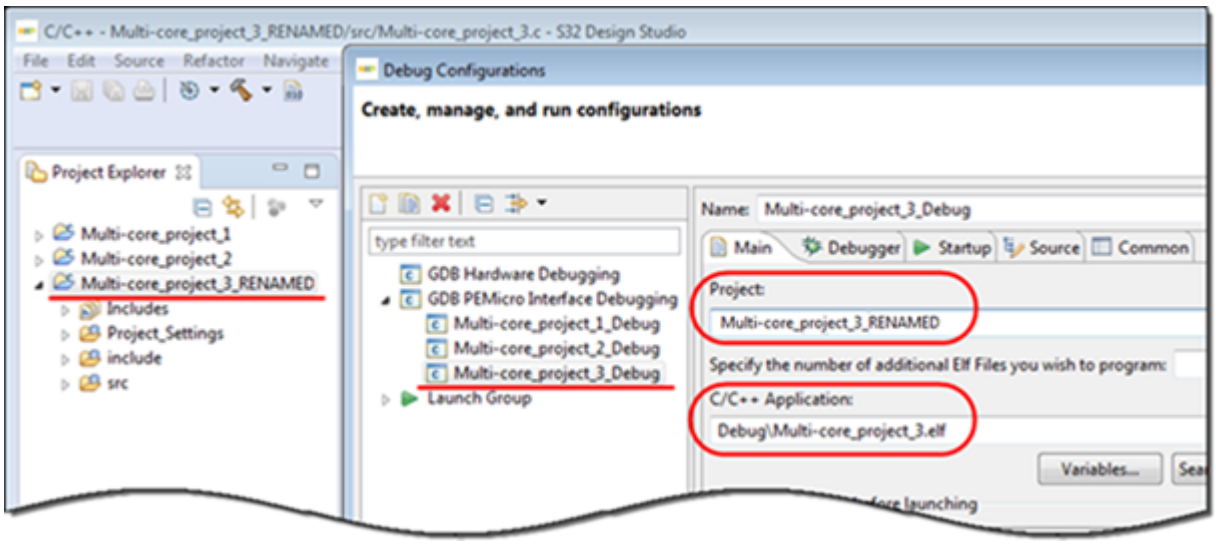
So if the project was renamed from existed project then some actions shall be done before debugging:

1. Uncheck the **Filter Deleted/Unavailable Projects** option from filter to display configurations:

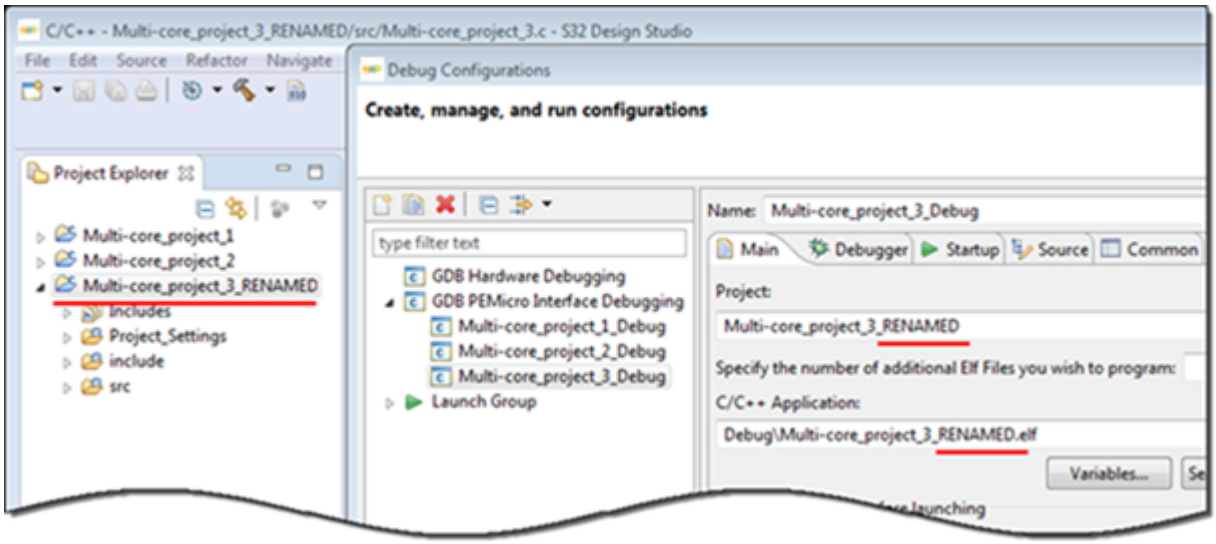


Debug Configurations dialog box

2. Select the configuration. The **Project** field is updated on the **Main** pane but the **C/C++ Application** field contain original name of the project:



3. Update the filename in the **C/C++ Application** field:

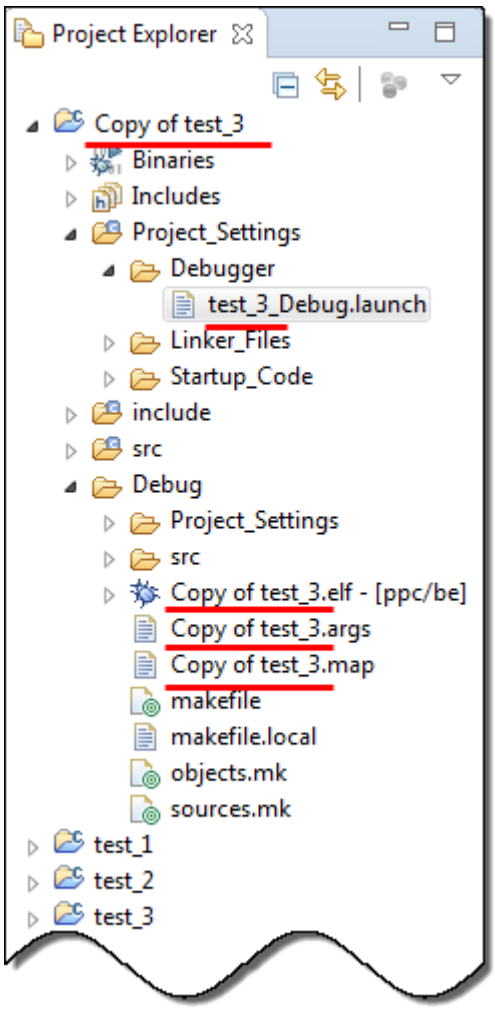


4. Click on the **Apply** button.

Now the launch configuration is ready and project can be debugged.

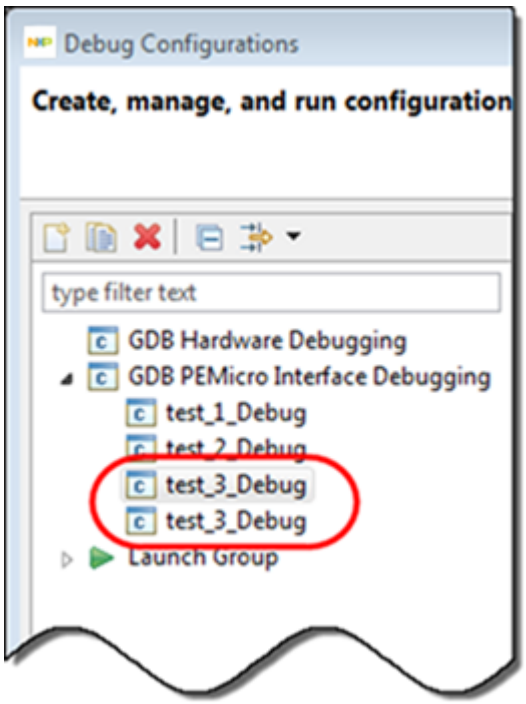
Prepare a copied project for debugging

For example, “**test_3**” project was copied to “**Copy of test_3**”:

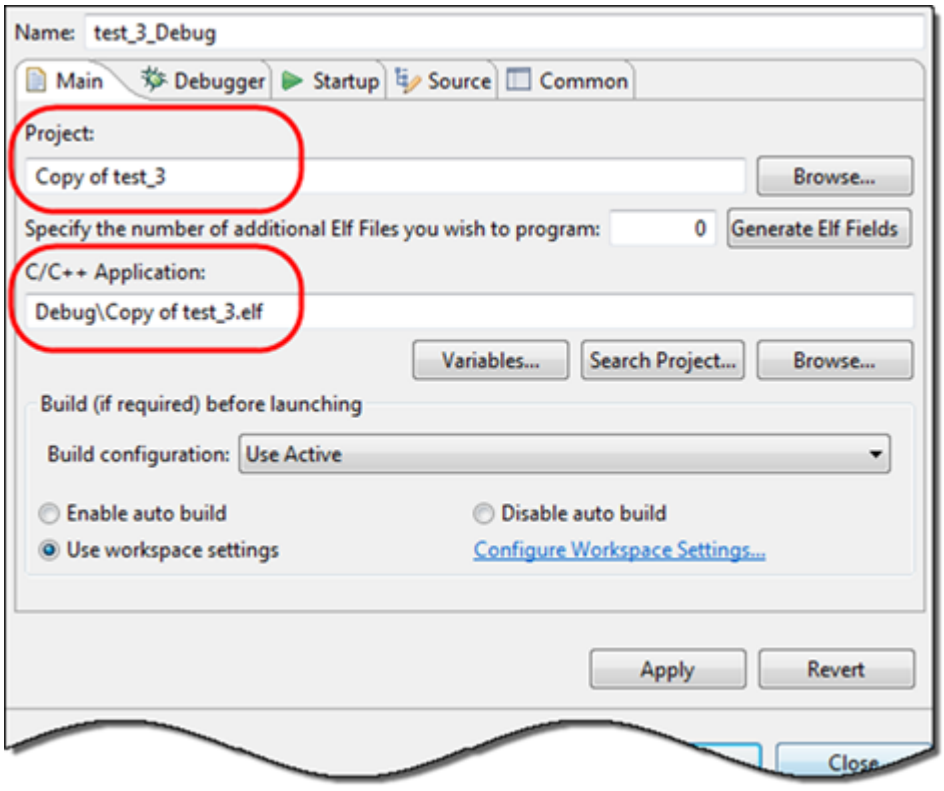


So if the project was copied from existed project then some actions shall be done before debugging:

1. Open **Debug Configurations** window for the new project "**Copy of test_3**".
2. Select the corresponding configuration:



3. Open the **Main** pane.
4. Rename **Project** value from "**test_3**" to "**Copy of test_3**" and **C/C++ Application** value from "**Debug\test_3.elf**" to "**Debug\Copy of test_3.elf**":



5. Click on the **Apply** button.

Now the launch configuration is ready and project can be debugged.

Chapter 5

Multicore debugging

Topics:

S32DS Power v2017.R1 allows to define multiple grouping of cores and perform multicore operations. Additionally, the chapter lists the steps to add multicore operations to S32DS Power v2017.R1 through the UI.

The debugger in S32DS Power v2017.R1 provides the facility to debug multiple processors using a single debug environment. The run control operations can be operated independently. A common debug kernel facilitates multicore, run control debug operations for examining and debugging the interaction of the software running on the different cores on the system.

Targeting core

The debugger connects to specific processor core through information provided in the **Debug Configurations** window. Specifically, the core index value on the **Main** tab of the **Debug Configurations** dialog box determines the core targeted for debug operations.

The core index value starts from “1”. That is, the first processor core has an index value of “1”, the second processor core has an index of “2”, and so on.

You can change this core index value on the **Main** tab > **Name** textbox, when you are modifying the settings of a Debug configuration.

Starting debugging session for core

To start debug for a specific core user need to connect the debugger to that core and start a debugging session. User can use the following methods:

- Start debug from **Debug Configurations** dialog box
- Start debug from **Run** menu
- Start debug from toolbar's **Debug** icon.

To start multicore debugging you should use the **Launch group** feature. First starts the main core, and then the sessions for the other cores start.

Debugging Specific Core

After you select the launch configuration and click **Debug**, the debugger downloads the program to the main core and the **Debug** perspective appears. Within the **Debug** view, the program’s thread appears. The thread is identified by its launch configuration name and the index value of the core that it executes on. If you are debugging source code, the program’s source appears in an Editor view.

To debug a specific core's program, click on its thread in the **Debug** view. The **Debug** perspective automatically displays the source, registers, and variables for this core. If you click on another thread, the **Debug** perspective updates all of the views to display that core's context.

Chapter 6

Connections

Topics:

This chapter describes the features and settings of the connections that interface the S32DS Power v2017.R1 debuggers with the target board.

For the IDE to communicate with the target hardware, you must specify several key items:

- connection type
- port over which debug communications is conducted
- target device name being debugged
- connection parameters.

The connection type determines what debugger protocol the debugger uses to communicate with the target. You can select the connection type in the **New S32DS Project** wizard (**New S32DS Project for <processor name>** step > **Debugger** list).

The others connection settings configure options specific for the hardware probe. After you make the option for the connection type you can enter connection settings using options in the **Debugger** panel of the **Debug Configurations** dialog box .

GDB PnE Micro connection

This chapter describes the features and settings of the PnE Micro connection that interfaces the debugger with the target board.

For the IDE to communicate with the target hardware, additionally you must select the interface type. The connection setting permits a connection to devices via following hardware interfaces:

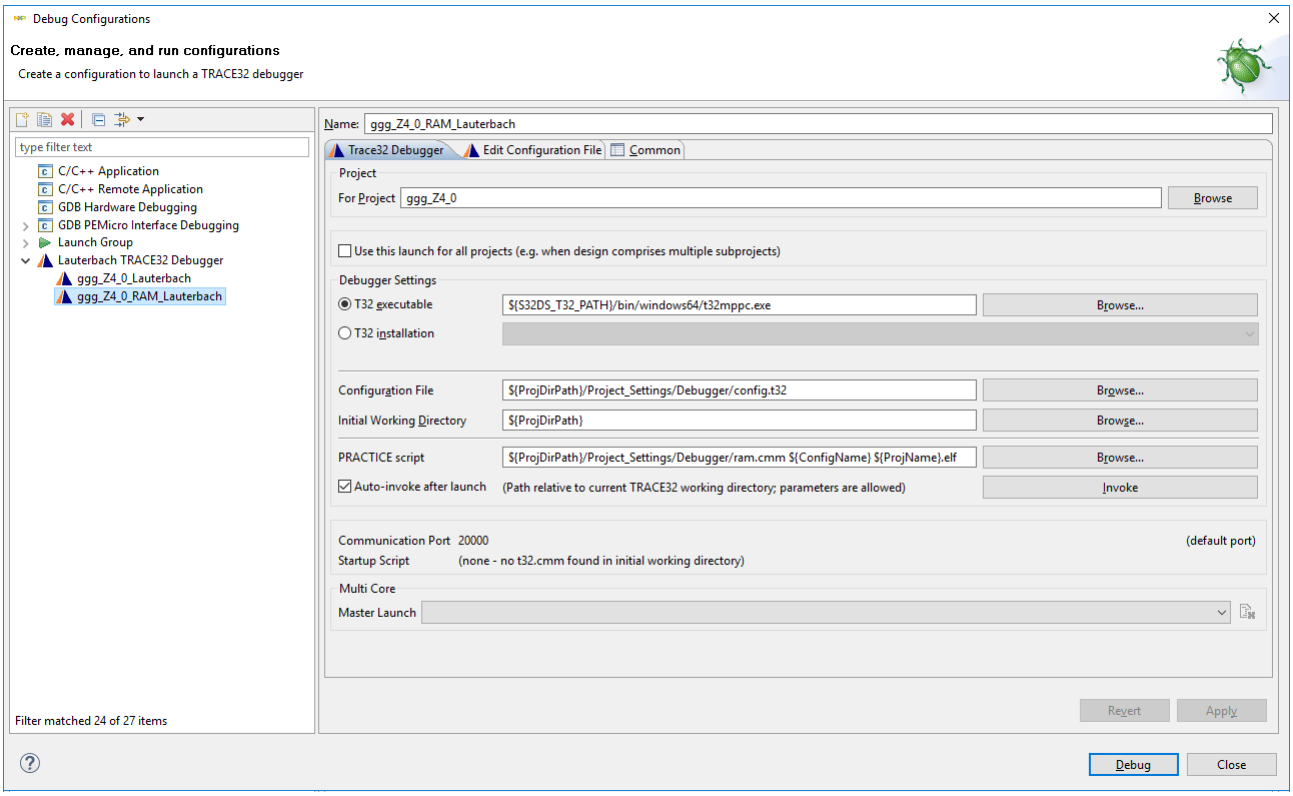
- USB Multilink
- Embedded OSBDM/OSJTAG
- Cyclone
- TraceLink
- OpenSDA Embedded Debug.

How to customize connection parameters for GDB PnE Micro interface see **PnE GDB Server Plug-In for Kinetis Devices**, **Debug Configuration User Guide** in the S32DS Power v2017.R1 Help (see **Freescale S32DS Power v2017.R1** > **Common Manuals** or the **Documentation** (the **Common Manuals** page)).

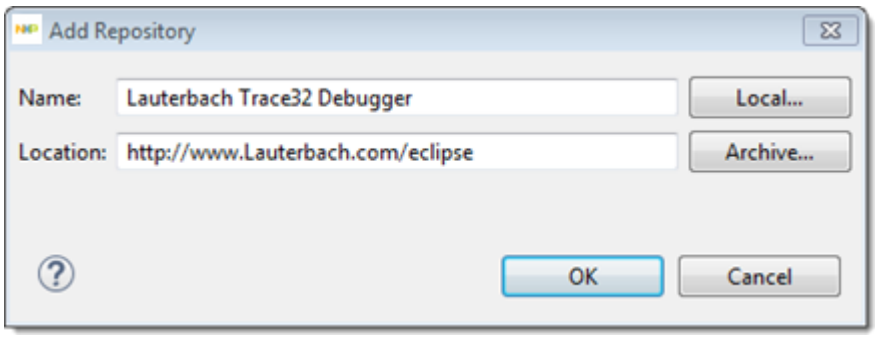
Lauterbach connection

This topic describes the features and settings of the Lauterbach connection that interfaces the debugger with the target board.

S32DS Power v2017.R1 supports the Lauterbach connection.



The Lauterbach TRACE32 Eclipse plug-in should be installed in the S32DS Power v2017.R1 to debug a project. Use the **Help** > **Install new software** menu to install it.



How to install the Lauterbach Trace32 Eclipse plug-in see the article Installing the Lauterbach Trace32 Eclipse plug-in software:

www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.ide.userguide%2Ftopic%2Fdebug_JTAGNeutInstTrace32pi_.html

How to customize connection parameters for Lauterbach see Lauterbach web site:

www.lauterbach.com

and the article Coupling for Eclipse:

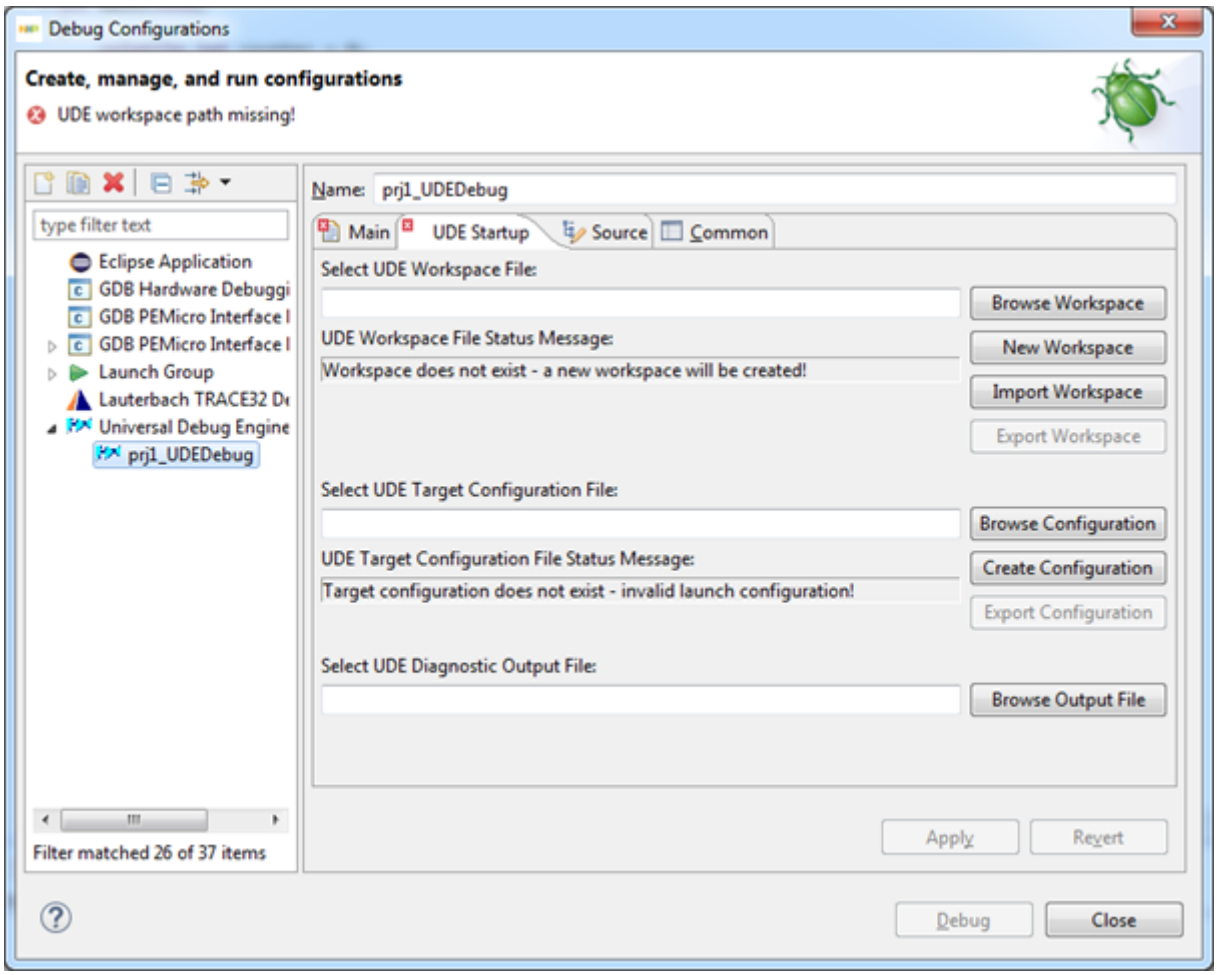
www2.lauterbach.com/pdf/int_eclipse.pdf

Universal Debug Engine connection

This topic describes the features and settings of the UDE connection that interfaces the debugger with the target board.

S32DS Power v2017.R1 supports the UDE connection.

The UDE Eclipse plug-in should be installed to debug a project).



The UDE Eclipse plug-in should be installed in the S32DS Power v2017.R1.

To use the UDE Eclipse plug-in:

1. Download Universal Debug Engine for Power Architecture from www.pls-mc.com
2. Install the UDE Eclipse plug-in (**<UDE install folder>\UDE 4.4\UDEEclipse4Integration.zip**) to S32DS Power v2017.R1 for PA. Use the **Help** > **Install new software** menu to install it.

After UDE Eclipse plug-in was installed user can select the Universal Debug Engine debugger in the **New S32DS Project** wizard.

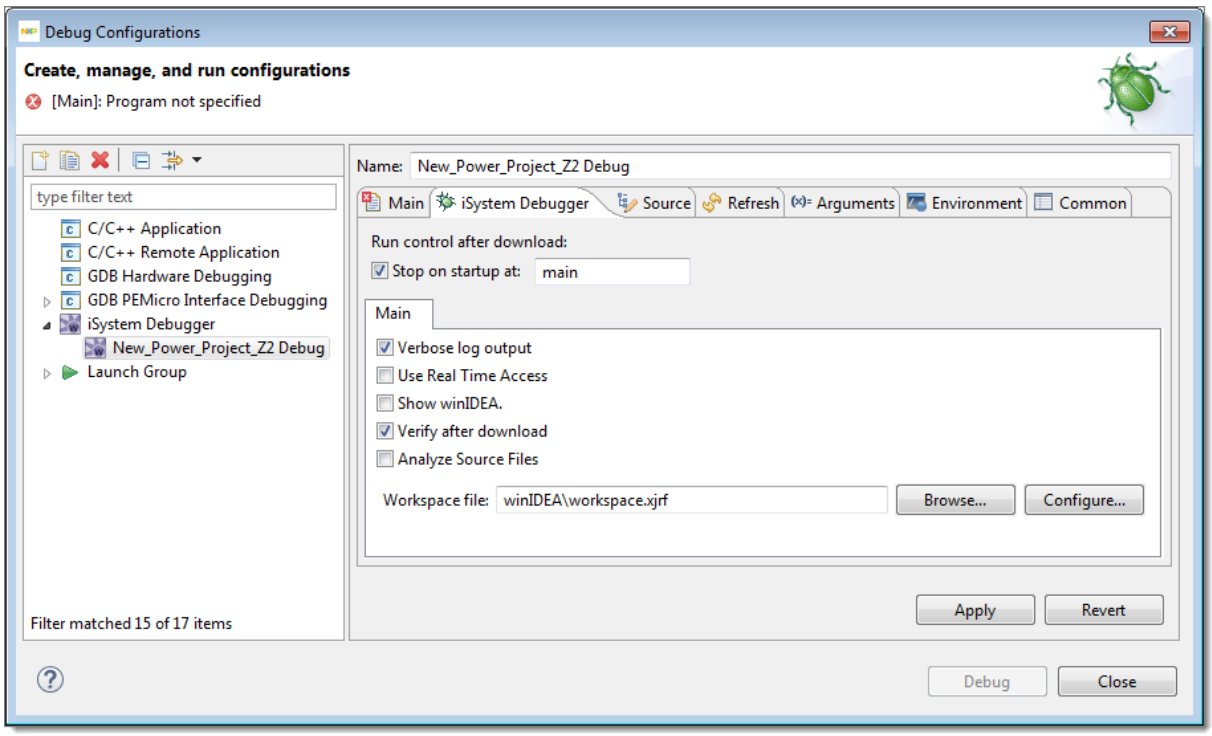
If the UDE debugger is selected then S32DS Power v2017.R1 creates debug configuration for the main core. User should set file names on the **UDE Startup** tab in the **UDE Workspace File** and **UDE Target Configuration File** fields. These files should be created in the UDE Visual Platform.

- See the UDE Reference Manual: **< UDE install folder>\UDE 4.4\Help**.
- How to install the UDE Eclipse plug-in see PLS web site: www.pls-mc.com
- How to customize connection parameters for UDE see PLS web site: www.pls-mc.com

iSystem connection

This chapter describes the features and settings of the iSystem module that interfaces the debugger with the target board.

S32DS Power v2017.R1 supports a target connection to iSystem.



The following software should be installed in the S32DS Power v2017.R1 to use the iSystem debugger:

- iSystem testIDEA version 9.12.273

  Download it from http://www.isystem.com/downloads/winIDEA/setup/winIDEA(x64)9_12_273.exe. Do not use the 9.12.256 version from the www.isystem.com site.
- iSystem Debugger Plug-in for Eclipse.

  Use the **Help** > **Install new software** menu to install the plug-in. Download it from the software site: http://www.isystem.si/eclipseUpdate/debuggerJuno42/ , accept the license agreement and restart IDE.

How to install the iSystem Debug Plug-in for Eclipse see the **Eclipse Plug-ins** web-page: http://www.isystem.com/download/eclipse

For details on the usage read the **iSystem Debug Plug-in for Eclipse User's Guide**:http://www.isystem.com/downloads/SDK/eclipse/iSystem-EclipseDebugPlugin-UsersGuide.pdf.

Chapter 7

Working with SDKs

Topics:

This chapter describes usage of Software Development Kits (SDKs) in S32DS Power v2017.R1.

SDK management

S32DS Power v2017.R1 supports using the following types of SDK in the development of products:

- predefined SDK
- custom user-created SDK

The predefined SDKs are delivered as plugins. The following ways of delivery the sources of the SDKs are supported:

- packaged in the plugin
- provided as directory in the S32DS Power v2017.R1
- link to GIT repository

SDKs delivered in the form of either source files or precompiled object files (libraries) can be integrated through the SDK manager. By using the SDK manager dialog you can define SDK parameters or the plugin (when using predefined SDKs) and add a user library. You can move a custom SDK between computers and reuse them in multiple installations of S32DS Power v2017.R1or its workspaces.

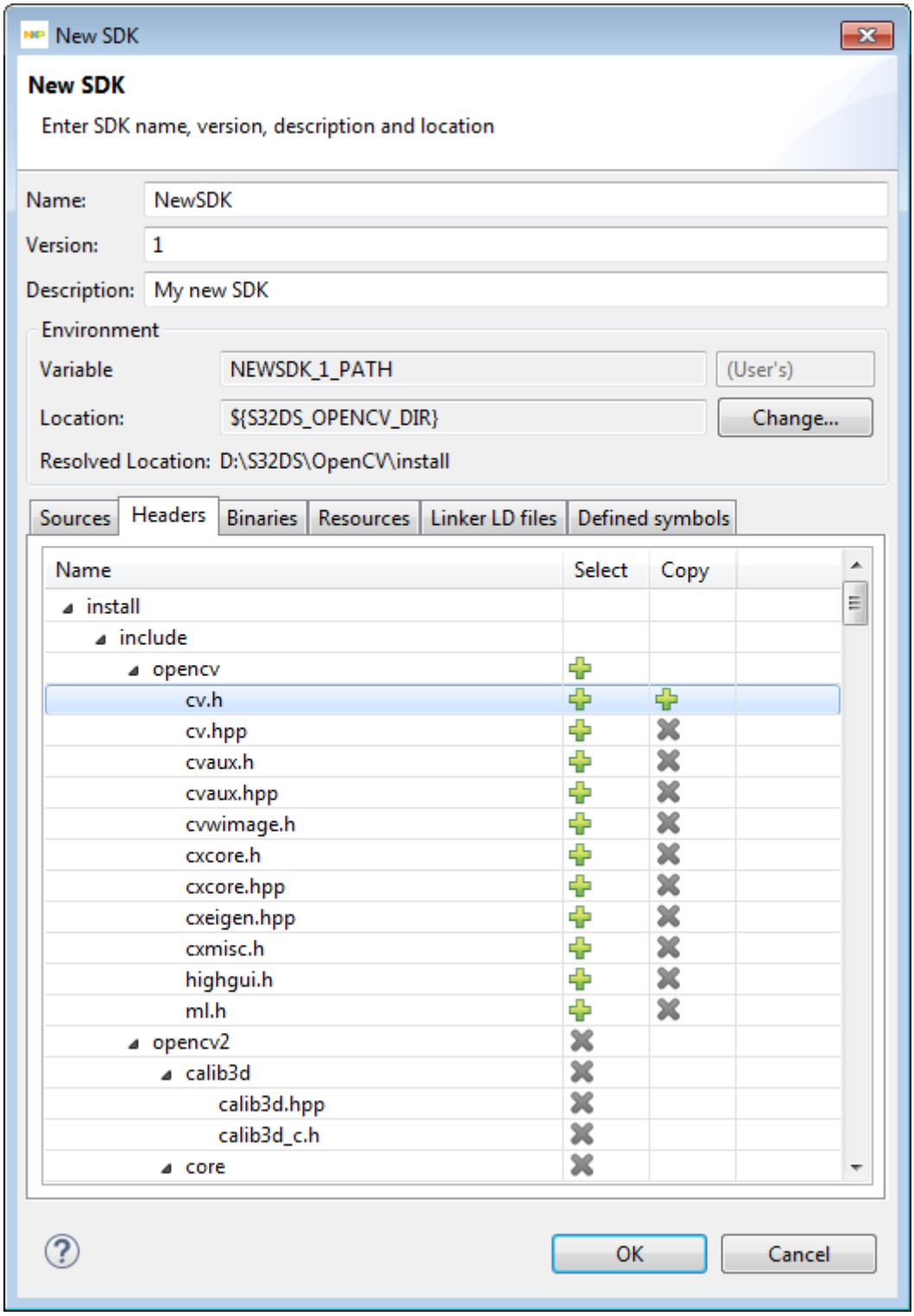
To use an SDK in your S32DS Power v2017.R1 project, add the SDK to this project. You can do that when creating a new project by using the **New S32DS Application Project** wizard, or you can add the SDK to an existing project by editing project settings. When a SDK added to project the SDK Explorer displays information about modules added to the project.

S32DS Power v2017.R1 performs the following actions when you add an SDK to the project:

- links configured source files to the project
- copies files to the project (if it is specified in configuration)
- specifies the location of include files in toolchain settings
- adds preprocessor symbols to toolchain settings
- adds reference to the library object file
- updates libraries (-L) paths in compiler and linker options.

If you detach SDK from project, all settings and linked files listed above are removed from the toolchain settings/ project. Only files that were specified as copied are stayed in project structure. You can manually remove the remnants from the project.

Following dialog window allows you to specify new or edit existing SDK settings:



The SDK manager provides the user with the following information about SDK:

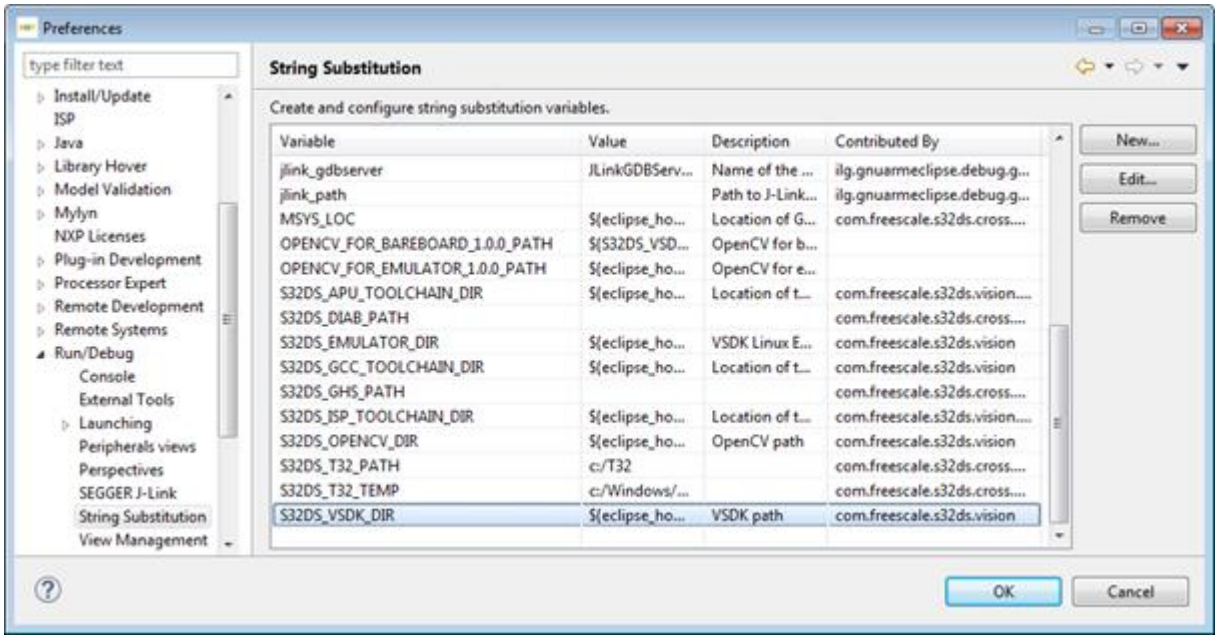
- name
- version
- description
- root directory
- list of header files
- list of source files

- list of library/object files
- list of other files
- information about symbols to be defined for preprocessor.

The information for the user defined SDKs can be edited, while for predefined (provided as plugins) –can be viewed only.

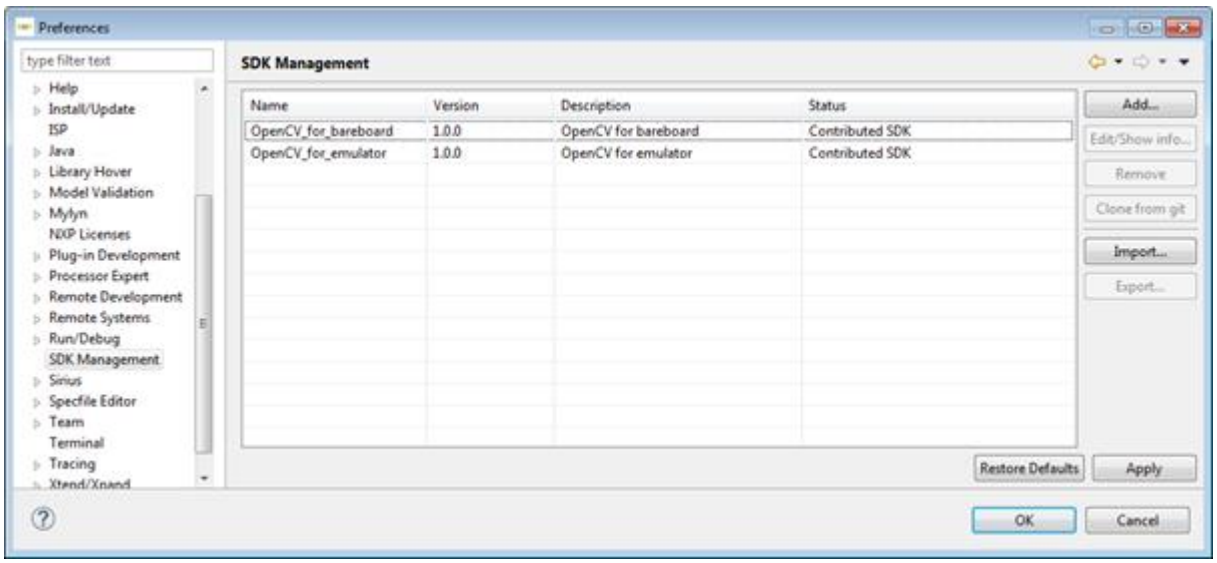
The list of files is created by scanning the root directory specified as location. The user can specify files to include to project by linking, or copying to the project.

SDK location can be edited using preferences page - **Run/Debug** - **String Substitution**:



SDK manager in workbench preferences

The **SDK management** page in global preferences allows to manage SDKs (add and edit or remove not built-in libraries) and import or export SDKs.

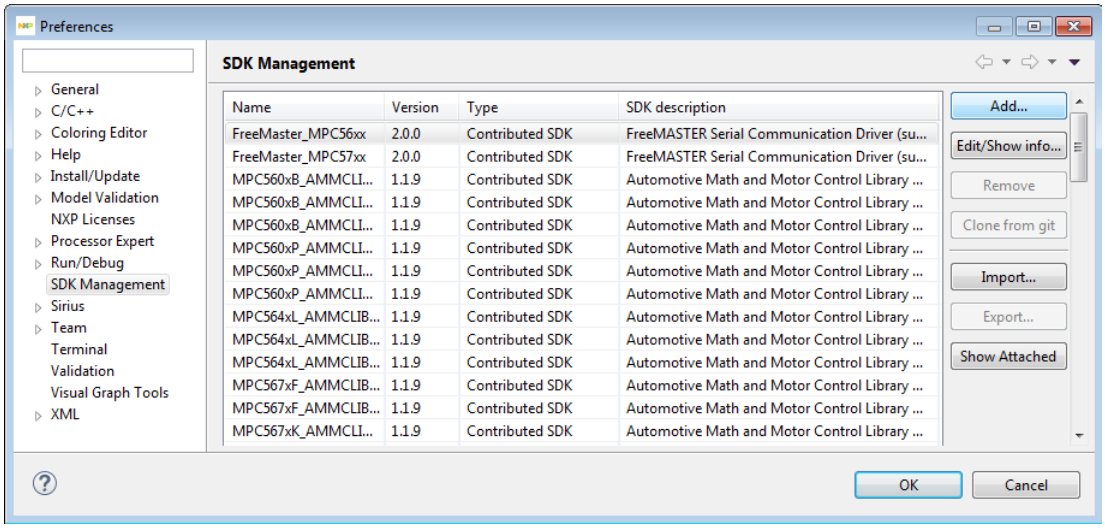


Adding custom SDK to Studio

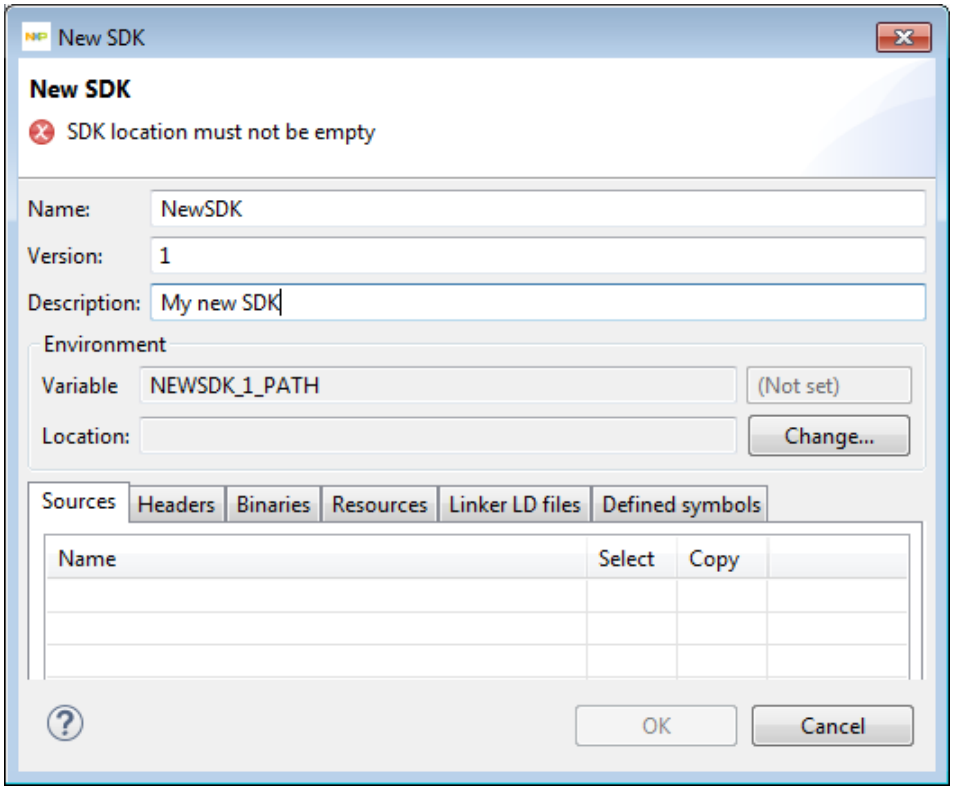
When you create a new project, you can only choose from existing SDK shipped with S32DS Power v2017.R1. If you want to add a custom SDK, you can edit S32DS Power v2017.R1 preferences and add the SDK files there. Files added in preferences are visible to all projects in the **Project Explorer** view.

To add a custom SDK:

1. In the main window, select **Window** > **Preferences** > **SDK Management** in the menu bar.
2. Click **Add...**.
   The **New SDK** dialog window opens.



3. Define Name, Version and Description of your new SDK:

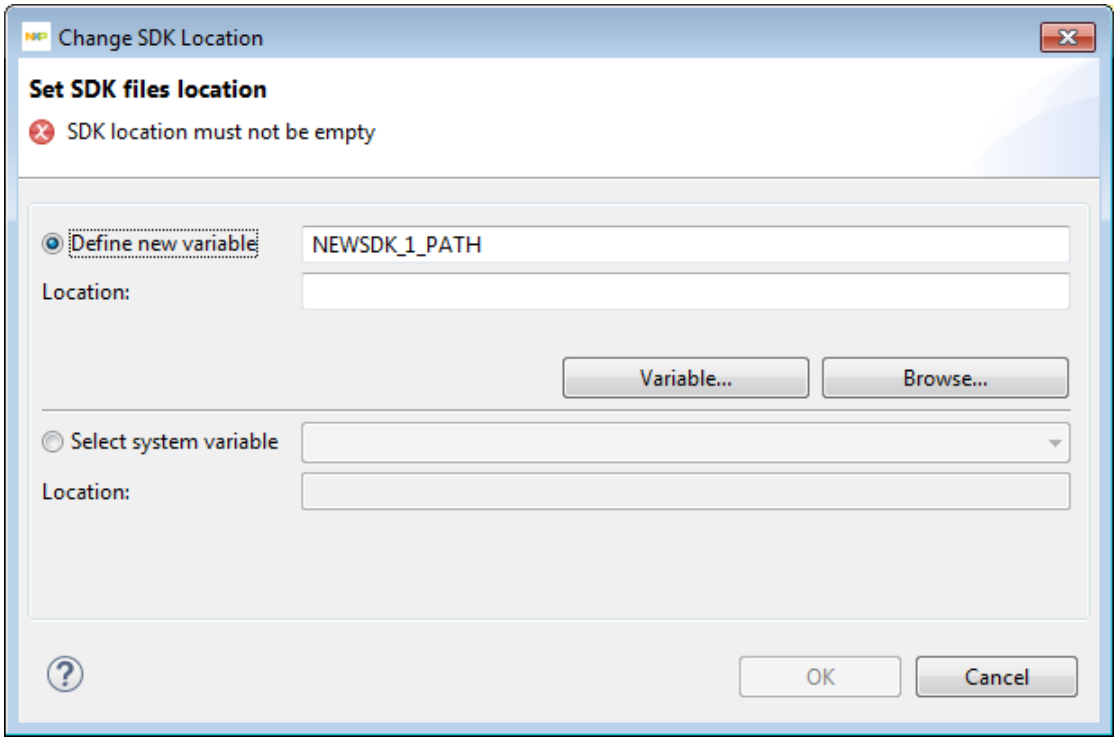


- **Name** - A valid C identifier. Must start with a letter. Allowed characters: letters, digits, and underscore.

- **Version** - Allowed characters: letters, digits, underscore, and period.
- **Description** (optional).

Note: Combination of name and version must be unique among other SDKs in the system because it is used for constructing Environment variable name.

4. To set location of environment variable click **Change...** button. You can either create new variable, use internal or system environment variable. Using internal variable allows you to share SDKs with other people or create SDKs to distribute widely.
5. In appeared Change SDK Location window you can Define new variable or Select system variable.

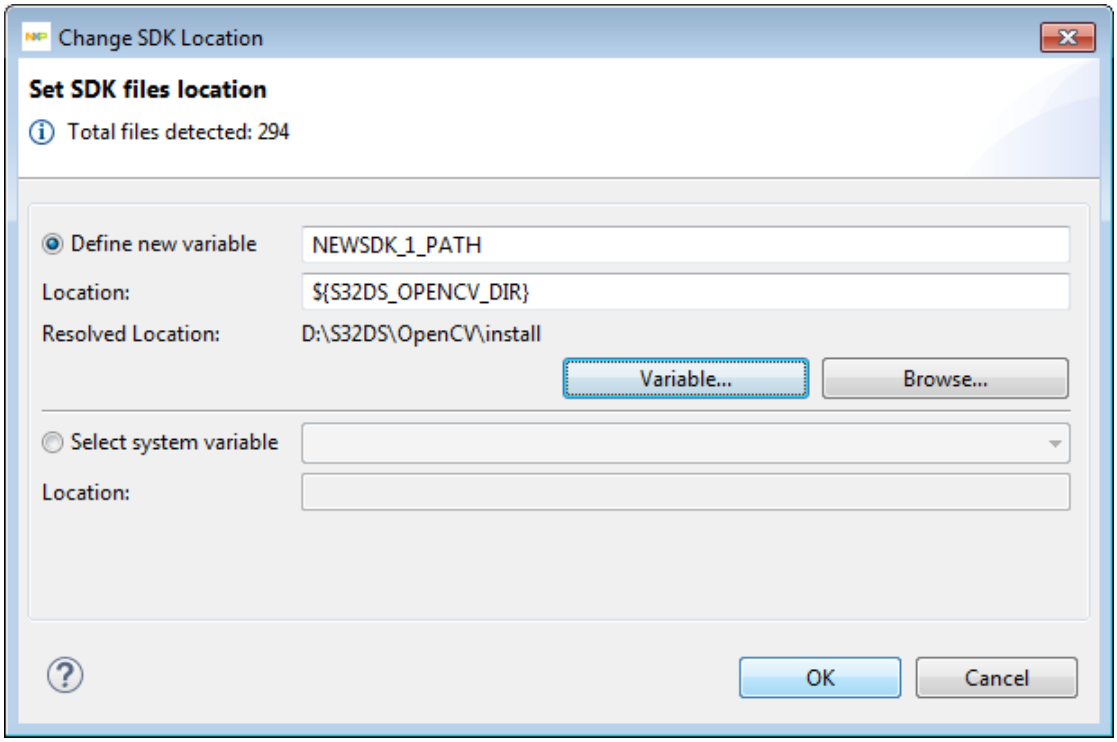


a) If you want to Define new variable, you should define the directory where SDK files are located by one of following ways:

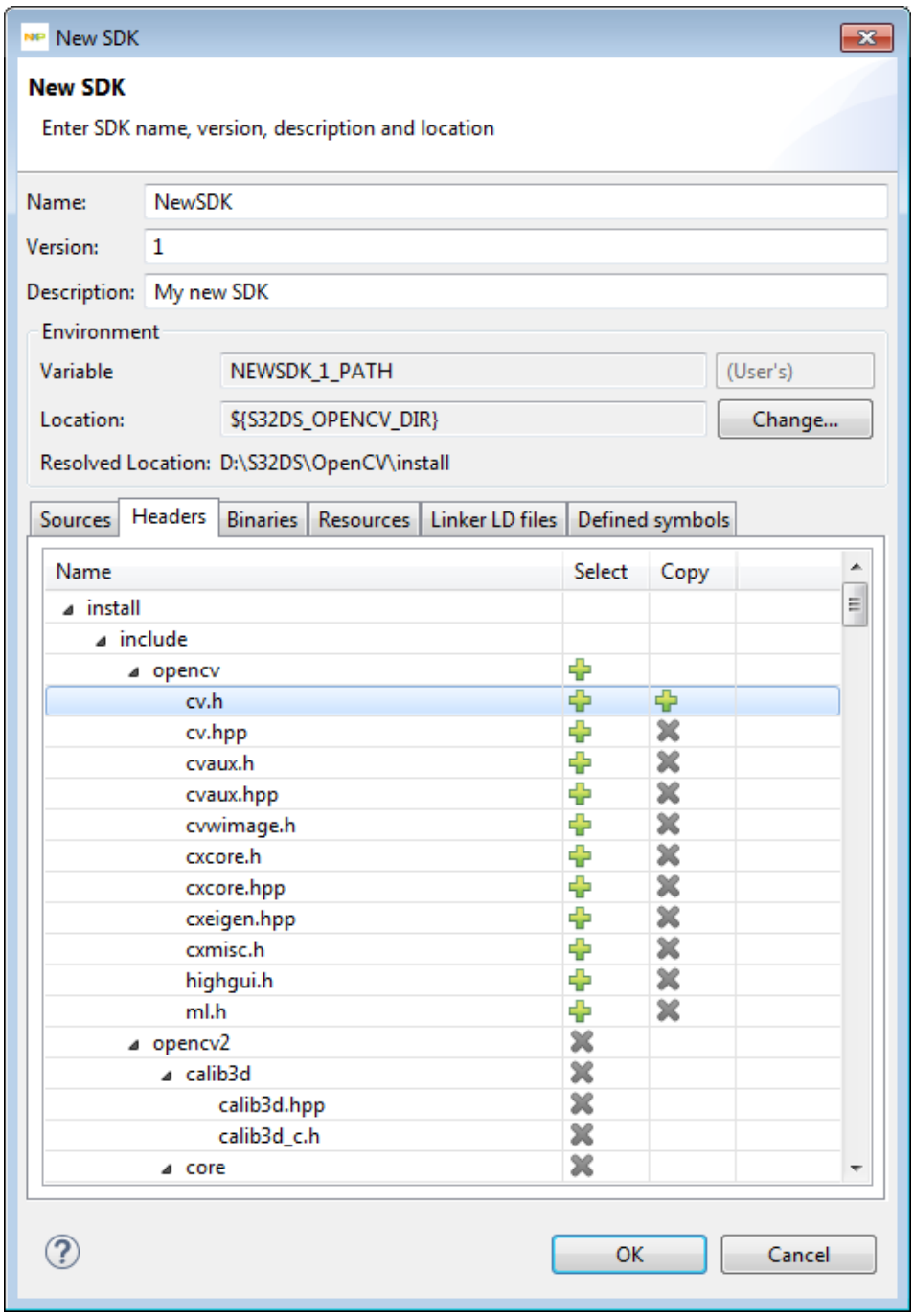
- Click **Browse...** and select folder where SDK files are located.
- Define path in Location field. Path may be defined as canonical file system path or as a reference to internal variable that points to SDK folder in file system.
- Click **Variable...** to use internal variable.

b) If you want to Select system variable, choose it from drop-down list.

After setting location of environment variable you can see Resolved location:



6. You can view and manage: Sources, Headers, Binaries, Resources and Linker LD files:



- Sources - usually **.c** and **.cpp** files.
- Headers - usually **.h** and **.hpp** files.
- Binaries
- Resources - any files (documents, images, etc.)
- Linker ID files - **.ld** files

In each files category you can:

- Mark with ✚ files in "Select" column. In this case files will be linked to the project during SDK attaching procedure.
- Mark with ✚ files in "Copy" column. In this case files will be copied to the project during SDK attaching procedure.

- default mark.

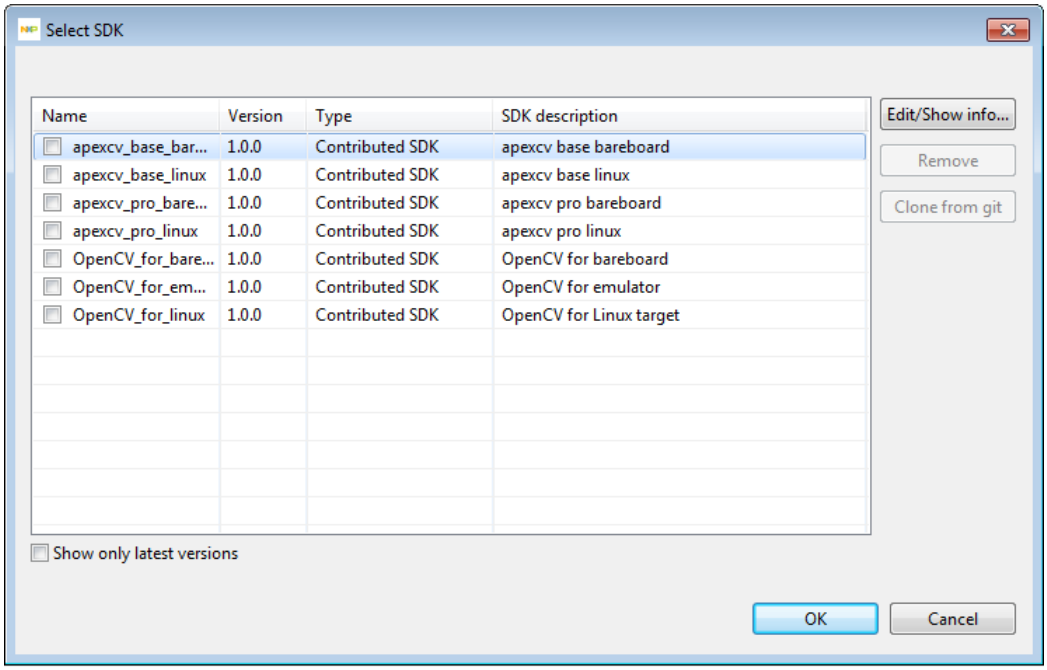
Note: When read-only (contributed/GIT) SDK is shown in this interface, its files are grouped not by extensions, but only according to SDK description. So, if “a.exe” is somehow defined as “<headerFile>”, it will be shown in Headers tab.

In Defined symbols tab you can view and manage information about symbols to be defined for preprocessor.

After SDK is created, descriptor of SDK is stored (in project properties for Project-local SDK or in Eclipse preferences for Global SDK) and SDK can be attached to project. New SDK is added to list of SDKs.

Selecting SDK in New S32DS Project wizard

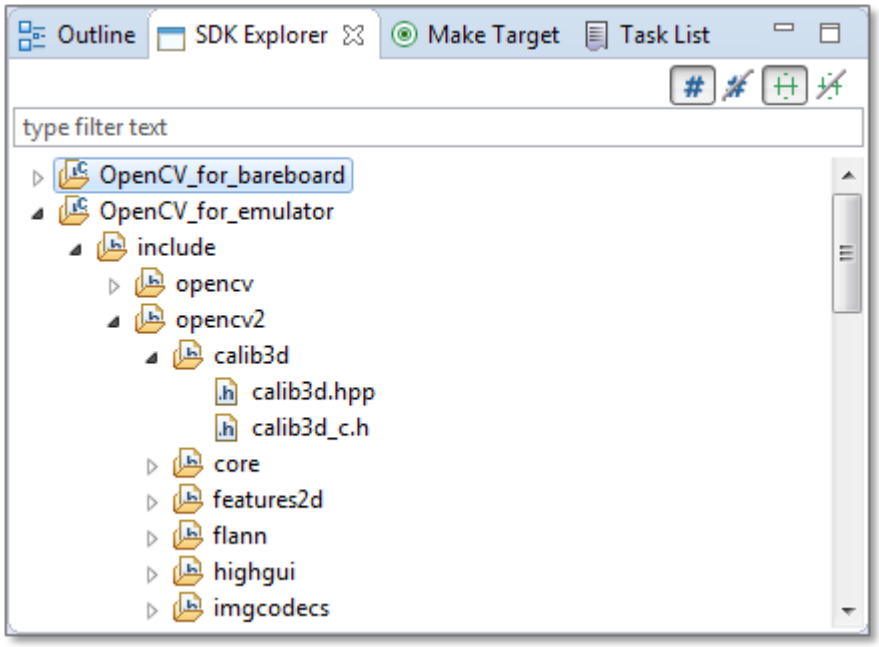
The second step of the **New S32DS Project** wizard contains a control **SDKs** to select SDKs for project. In the **Select SDK** dialog user can select SDKs to attach to newly created project.



The set of SDKs depends on the selected processor/ toolchain/ core, so only modules intended for the device should be available. The user can include any SDK – predefined (**Contributed SDK**) or user’s created (**Local**) - into a project.

SDK Explorer

The **SDK Explorer** view shows SDKs attached to a project selected in the **Project Explorer**. **SDK Explorer** view displays information from the header files with defines and functions.



User can use the drag-and-drop possibility to drag a function from **SDK Explorer** into a source code in editor. When a function is dragged to a source file the **SDK Explorer** inserts the **include** statement for corresponding header file at the beginning of the file and adds the function call to the source with names at the place of parameters. If the function has return – it is written as **return_type = function_name (first_param_type, second_param_type)**;

For example for the function with prototype:

flexcan_status_t FLEXCAN_HAL_Enable(CAN_Type * base)

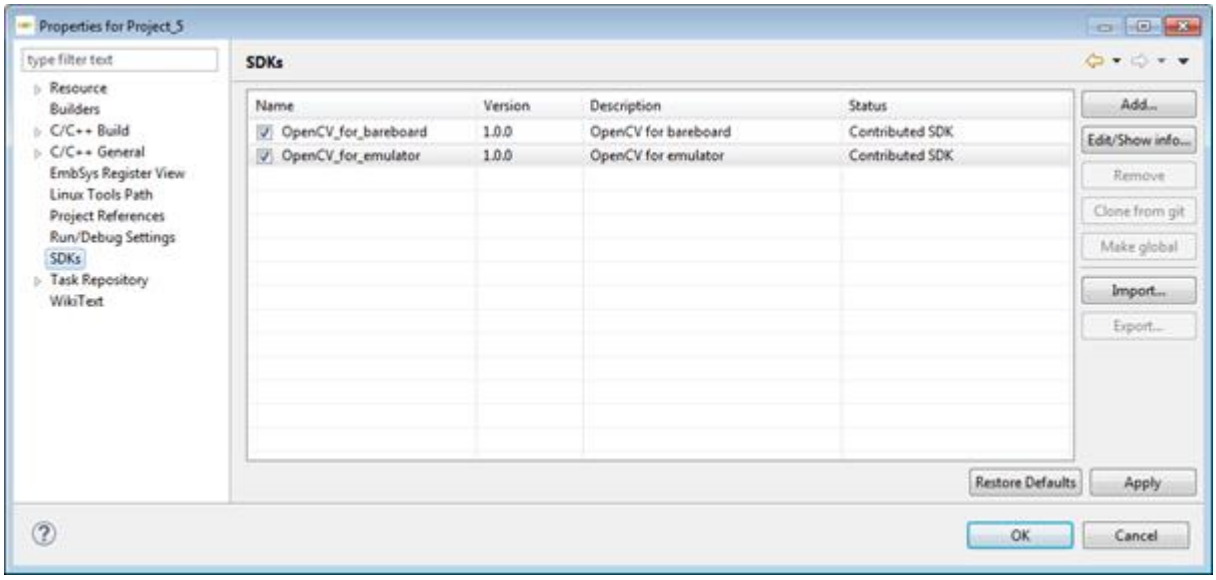
the following statement is added to source file (when it is dropped):

flexcan_status_t = FLEXCAN_HAL_Enable(CAN_Type *);

When a **"define"** is dragged to the source file – it is added as is, without closing semicolon.

SDKs property page

SDKs property page in a project properties allows to attach\detach SDKs to the project. Include (-I) and libraries (-L) paths are automatically updated in compiler and linker options according to currently attached libraries:



The SDKs project property page displays information about all compatible SDKs which are defined within S32DS Power v2017.R1 – either installed with plugin (preconfigured) or user's defined.

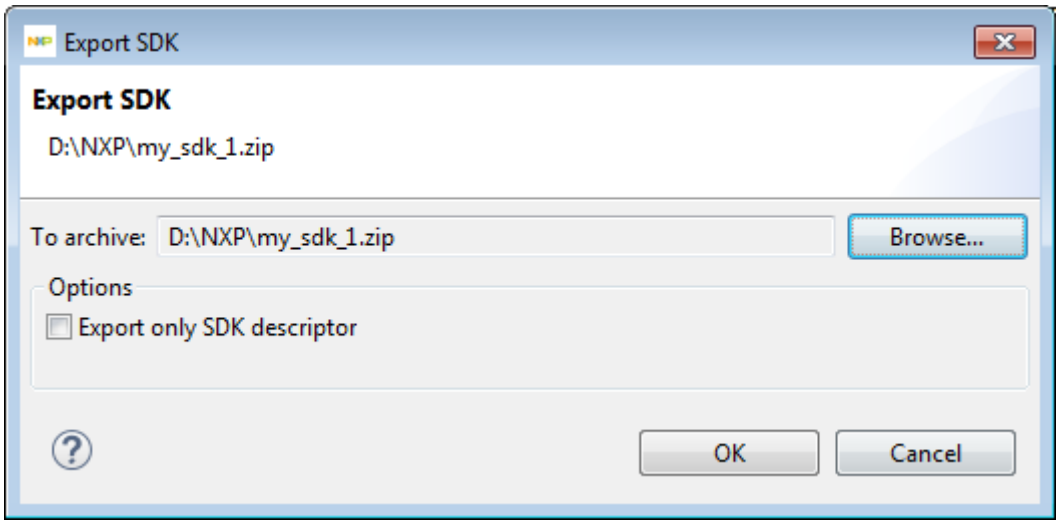
The SDKs list is filtered according the following criteria:

- supported compiler(s)
- supported language (C only or C/C++)
- supported architecture/core – if the SW module is independent from peripherals and depends only on core type
- supported device (core) – if SW module use some HW modules, then it could be used only for certain device (or core in case of multicores on this device).

Filtering can be switched off and user can select SDK by his choice.

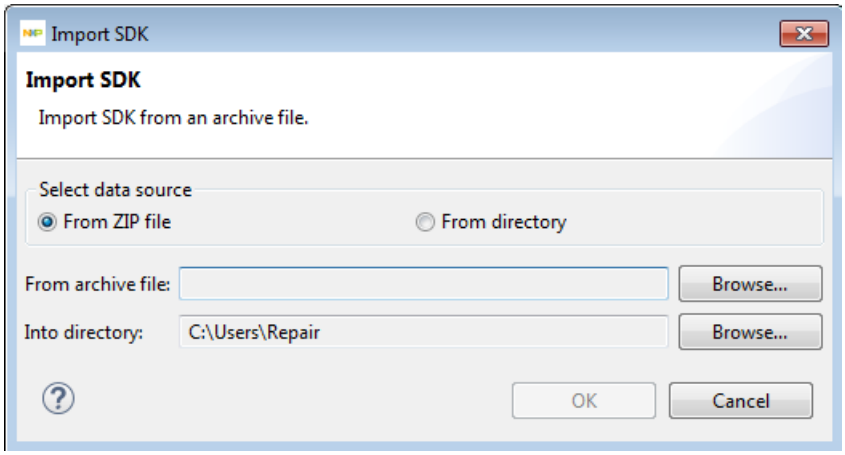
Import/export

User can export user-defined local SDK using the **Export…** button on the SDKs property page:



Enter or select the path to archive where to store SDK content or only xml-descriptor if the appropriate option is selected (the **Export only SDK descriptor** checkbox is selected).

Also SDK import is available from previously created archive file:



User defined SDK can be transferred. The predefined SDKs should be properly installed on different S32DS installation.

When SDK is exported the user might have possibility to include both SDK configuration file (XML-descriptor) as well as the header/source/library files, or only configuration file. The export creates zip-file.

When the SDK is imported from zip-file it requests location of header/source/library files:

- to link them (if only configuration was exported)
- to place them (if the whole SDK was included into export).

SDK delivered through GIT repository

The predefined SDK could have sources delivered through GIT repository, then instead of the files packaged in plugin or references in product layout a provided URL for GIT repository can be used as well as label. When the user start using this SDK he creates local copy of repository, defines location and provide credentials for GIT. The user cannot edit URL or label.

In case of GIT repository using the definition of file set and options is provided via manifest file. Manifest file is defined in the plugin configuration.

Chapter 8

SPT Graphical Tool Reference Manual

Topics:

SPT Graphical Tool Introduction

This document describes how to create a SPT programs as the graphical models using the S32 Design Studio SPT Graphical Tool and is targeted to end users.

This chapter presents an overview of the manual and introduces you to the information layout of the manual.

The topics in this chapter are:

Quick Links	Lists web-links to S32 Design Studio pages
Terminology	Lists terms used in the document
About this manual	Describes the contents of this manual
Accompanying documentation	Describes supplementary SPT Graphical Tool documentation, third-party documentation, and references

S32 Design Studio SPT Graphical Tool allows you to use graphical modeling workbench by leveraging the Eclipse Modeling technologies. It provides a workbench for model-based architecture engineering. Graphical Tool equips teams who have to deal with complex architectures. The Graphical Tool includes everything necessary to easily create and manipulate models. The output of this tool is SPT assembler source code.

The SPT Graphical Tool is integrated with the New S32DS Project wizard for devices which have SPT module. SPT1, SPT2 and SPT2.5 versions of SPT modules are supported. The SPT Graphical Tool can be used after project creation with the New Graph Tools Project wizard. For detailed description of new project wizards see **S32 Design Studio for Power Architecture, Version 2017.R1 Reference Manual**.

Quick links

- S32 Design Studio page (overview, downloads) www.nxp.com/S32DS
- S32 Design Studio community (for publicly shared cases) community.nxp.com/community/s32/s32ds
- Technical support (for confidential issues) www.nxp.com/support Hardware and Software link

Terminology

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

Table 47: Terminology

Term	Description
IDE	an integrated development environment
modeling project	a special kind of project in your workspace which makes it easy to manipulate representation files and semantic models in a consistent way
representation	a particular diagram which you created on your semantic model. It is simply a more general term than «diagram»

Term	Description
representation file	a file in which Graphical Tool stores information related to which representations you created, what appears on them, the positions and colors of the elements, etc. These files have a .aird extension (typically <project name>.aird). Representation files reference the semantic model(s) they contain representations for, but you semantic models are kept unaware (and unpolluted) of any Graphical Tool-specific data
resource	a file (in your workspace or inside a plug-in) which contains a model
<keyword keyref="product-name"/>	the <keyword keyref="product-fullname"/>
semantic model	a model (or models) which contains your data. It can be stored in one of several resources (files) which can reference each other
viewpoint	a set of representation descriptions which provide a specific point of view on some kind of semantic model

About this manual

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

Table 48: Manual contents

Chapter / Appendix	Description
Introduction	This chapter
Modeling Projects and representations	Explains how to use the SPT Graph Tool to create and work with SPT Graph Tool Projects
Diagram editor	Explains how to use the diagram editor (modeler) supported by SPT Graphical Tool which allow to view and manipulate data in a graphical way
Preferences	Explains how to use available preferences and configuration parameters to customize SPT Graphical Tool

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

Accompanying documentation

S32DS Power v2017.R1 includes an extensive documentation library of user guides, reference manuals etc. Take advantage of this library to learn how to efficiently develop software using the S32DS Power v2017.R1 programming environment.

S32DS Power v2017.R1 documentation presents information in the following formats:

- **PDF** Portable Document Format of the manuals, such as the Common Features Guide, Reference Manual or Release Notes
- **HTML** Hypertext Markup Language version of the manuals

PDF-documentation

The **S32 Design Studio Documentation Suite** includes the S32DS Power v2017.R1 PDF-documentation.

You can access the manuals, FAQ, etc. by:

- opening the **start_here.html** in **<S32 Design Studio install dir>/S32DS/help** directory, where **S32 Design Studio Install dir** is the directory that S32 Design Studio was installed into
- selecting **Help** > **Documentation** from the S32DS Power v2017.R1 menu bar
- selecting the **Documentation** shortcut.

Online help

To view the online help:

1. Select **Help** > **Help Contents** from the S32DS Power v2017.R1 menu bar.
2. Select required manual from the **Contents** list.

Release notes

Before using the S32DS Power v2017.R1, read the developer notes. These notes contain important information about last-minute changes, late-breaking information about new features, bug fixes, incompatible elements, known problems, or other topics that may not be included in this manual.

Read the Release notes in this directory:

<S32 Design Studio install dir>/S32DS/Release_Notes.

The release notes for specific components of the S32DS Power v2017.R1 are located in the **Release_Notes** directory in the S32DS Power v2017.R1 installation directory.

Modeling Projects and representations

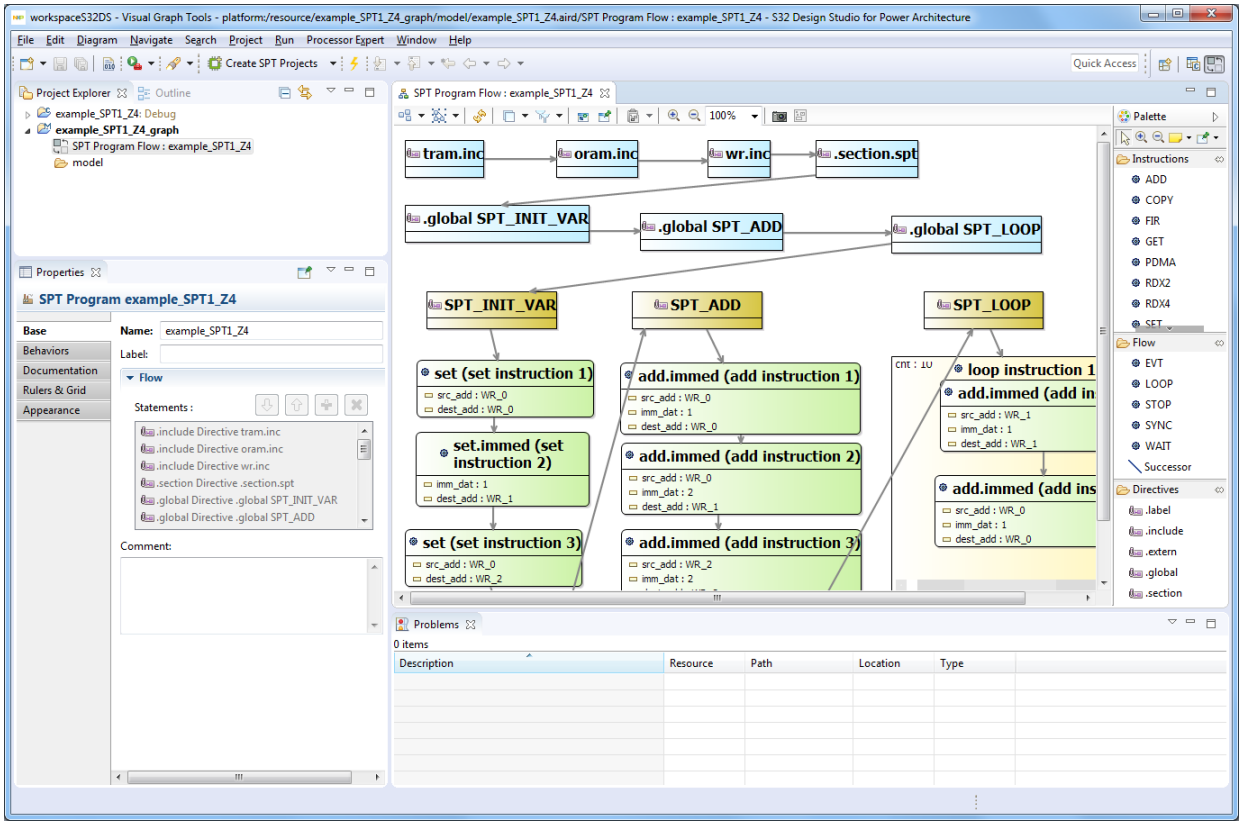
This chapter explains how to use the SPT Graph Tool to create and work with SPT Graph Tool Projects.

Graph Tools perspective

When a new Graph Tools Project was created, it opens on the **Graph Tools** perspective. This perspective provides all the required views and menus.

The Graph Tools perspective provides the following views by default:

Project Explorer	The main UI to interact with your models.
Properties	The view gives detailed information about the currently selected element. Depending on the nature of the selected element, some of these properties can be edited directly in the Properties view with immediate effect.
Problems	The view contains information markers of different severities (information only, warnings, or errors). This is where you will find validation errors on your models for example.

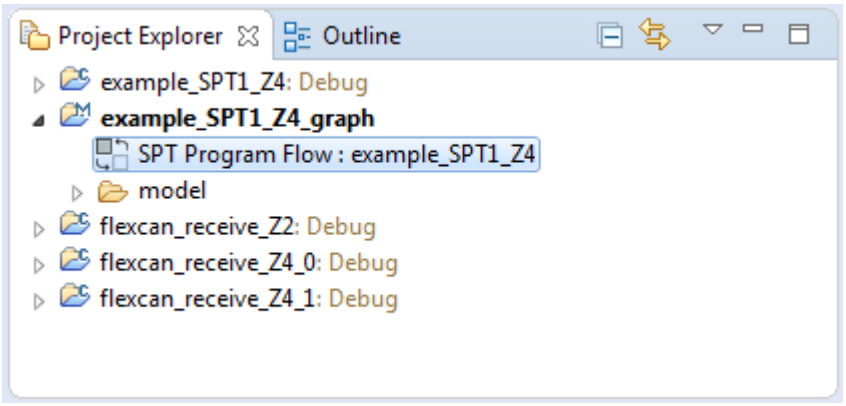


This perspective also provides useful actions available by right-clicking in the views.

This perspective can be customized by adding, moving or removing views, shortcuts, etc. For example the **Outline** view can provide a structural overview of the model currently opened. It shows a miniature view of the whole diagram on which you can easily navigate to other parts of the diagram for large ones.

Project Explorer view

The **Project Explore**r view shows all the projects in you workspace and the files they contain. It adds some special capabilities to modeling projects, to allow you to view and manipulate your semantic models and their diagrams inside the **Project Explorer**.



Representations files which are part of a modeling project or model files are hidden and non-available in the **Project Explorer**. They can't be expanded to display their content directly inside the **Project Explorer**.

The **Project Explorer** supports the «Link with Editor» feature, which can be enabled by pressing the icon in the top right corner of the view (the one with two horizontal arrows, pressed in the screenshot above). When this mode is enabled, if you have a representation opened, clicking anywhere on it will automatically select the corresponding

representation inside the **Project Explorer** (expanding the project and files if necessary). Conversely, if you select a semantic element from one of your semantic models in the **Project Explorer** view and if this element is represented somewhere on the opened editor, it will be automatically selected. This can be very useful when you have many projects and representation or large representations to avoid getting lost.

Modeling Projects

Modeling Projects are used in SPT Graphical Tool to organize and manage your models and their diagrams.

A Modeling Project is created by the **New Graph Tools Project** wizard. For further information, see **S32 Design Studio for Power Architecture, Version 2017.R1 Quick Start Guide**.

After finishing the wizard, the **Graph Tools** perspective opens and modeling project appears in the **Project Explorer** view. By default the view contains the following elements:

- **SPT Program Flow: <Graph Tools project name>** object - diagram of created Graph Tools project.
- **model** folder - representation of created Graph Tools project properties (It contains in file system the **<Graph Tools project name>.aird** file, which is the top-level representation file for the Graph Tools project. **Aird**-file is hidden and non-available in the Project Explorer).

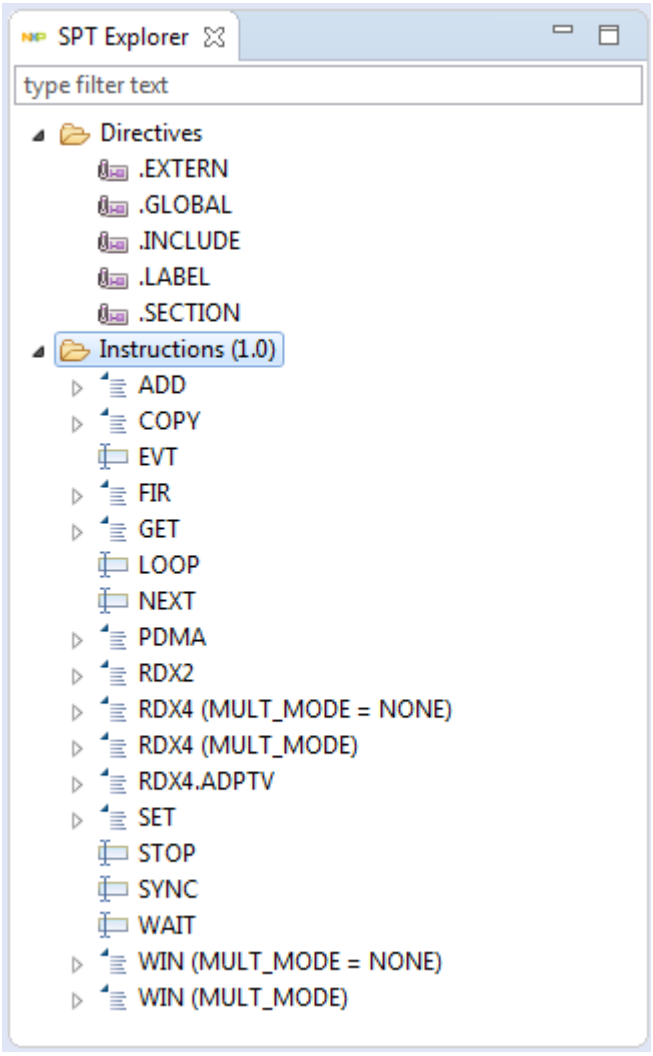
Although semantic data can be stored within models, modeling projects can also store semantic data of your models. The stored data is used to display diagrams. **Modeling projects** store representation data for diagrams in AIRD representation files. These representation files are created by the **New Graph Tools Project** wizard.

Modeling projects provide actions to manage viewpoints and representations: viewpoints are associated with modeling projects, so the viewpoints that will be available for representations of a given Modeling Project are attached to this project.

Modifying SPT source code

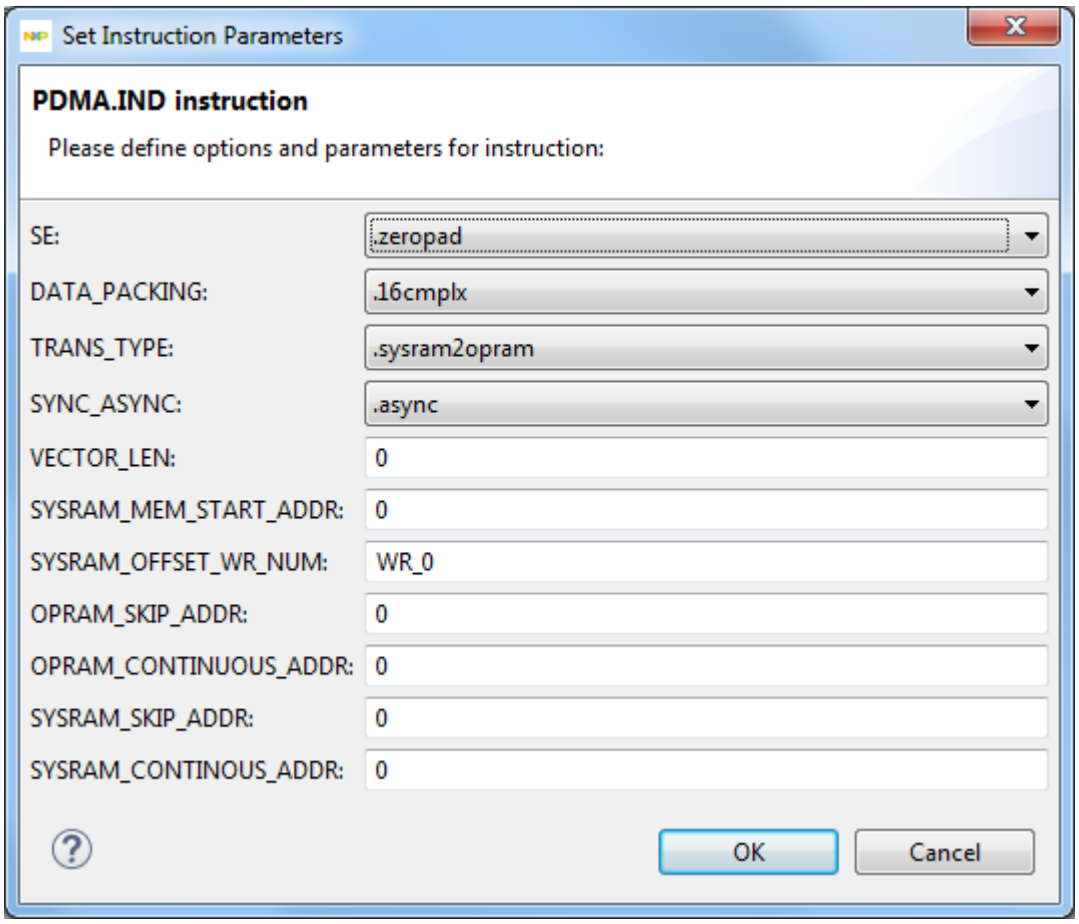
The **SPT Explorer** view is designed for modifying SPT assembler source code, therefore it shows all Instructions, Flow and Directives sets according to SPT version of the project. To add this view to perspective choose **Window** > **Show View** > **Other...** > **Other** > **SPT Explorer** from the S32DS Power v2017.R1 menu bar.

The **SPT Explorer** view displays elements in a tree format when <project_name>.spt file is chosen in spt_gen folder of the project in the **Project Explorer** view or opened in the editor area (for detailed description of generating the source code of SPT assembler from the diagram see SPT generating). To display the content of a particular category, expand the tree element of the category of interest.



You can modify your SPT assembler source code using **SPT Explorer**. For example, to add PDMA instruction:

1. Expand Instructions node in the **SPT Explorer** view and find PDMA element.
2. Expand PDMA and choose one of the two possible options (PDMA or PDMA.IND).
3. Drag-and-drop it into the code in editor. **Set Instruction Parameters** window appears.
4. Define provided options and parameters and click **OK**.



As a result, the following statement is added to <project_name>.spt file:

```
pdma.ind .zeropad .16cmplx .sysram2opram .async 0 0, WR_0, 0, 0, 0, 0
```

In a similar way can be added each directive, instruction or flow element.

Diagram editor

This chapter explains how to use the diagram editor (modeler) supported by SPT Graphical Tool which allows to view and manipulate data in a graphical way.

Introduction

Graphical Tool provides support for diagrammatic representations, which represent information in a graphical way. The Graphical Tool is used to create, visualize and edit your models using interactive editor - graphical modeler. User uses preconfigured modeler specially adapted to his needs. The modeler does not require any programming to create a model.

A diagram editor is divided in three areas:

- the main graphical area, which shows the elements and supports direct interaction with them
- the palette, which gives access to creation tools to add new elements to the diagram
- the toolbar at the top of the graphical area, which provides additional, more global operations.

In addition to these areas which are immediately visible inside the diagram editor, it is also possible to interact with the diagram and its elements through the **Properties** view and through contextual menus available on the graphical

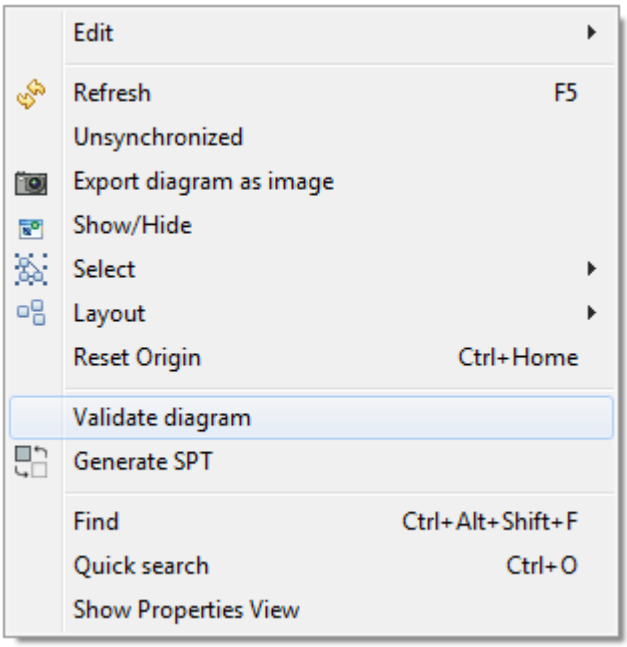
elements themselves and on the diagram's background. The **Outline** view shows a graphical overview of the diagram which can be used to navigate inside it if the whole diagram is not visible in the main area.

Main actions

Validating the model

On a diagram, you have validation rules. These rules were previously created. As a user of the modeler, you can validate these rules on your model.

To validate the rules open the context menu of the diagram itself (by right-clicking on its background) and choose the **Validate diagram** action.

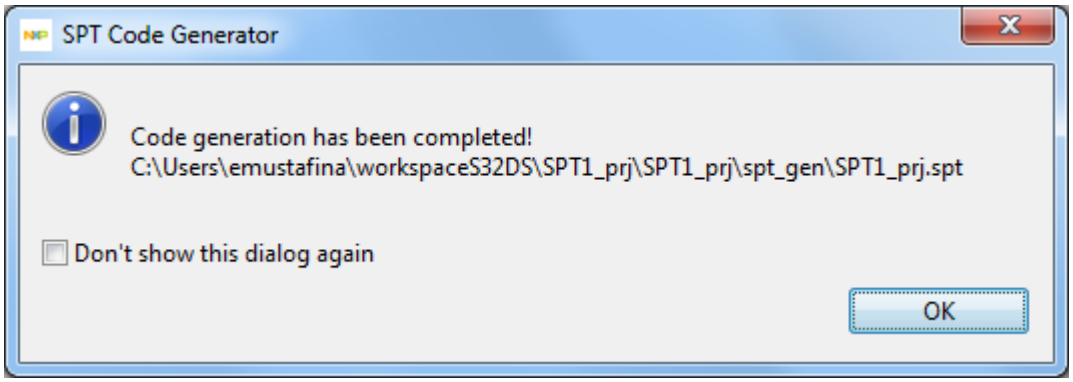


If a problem was found during model validation then the **Problems** view displays the error or warning.

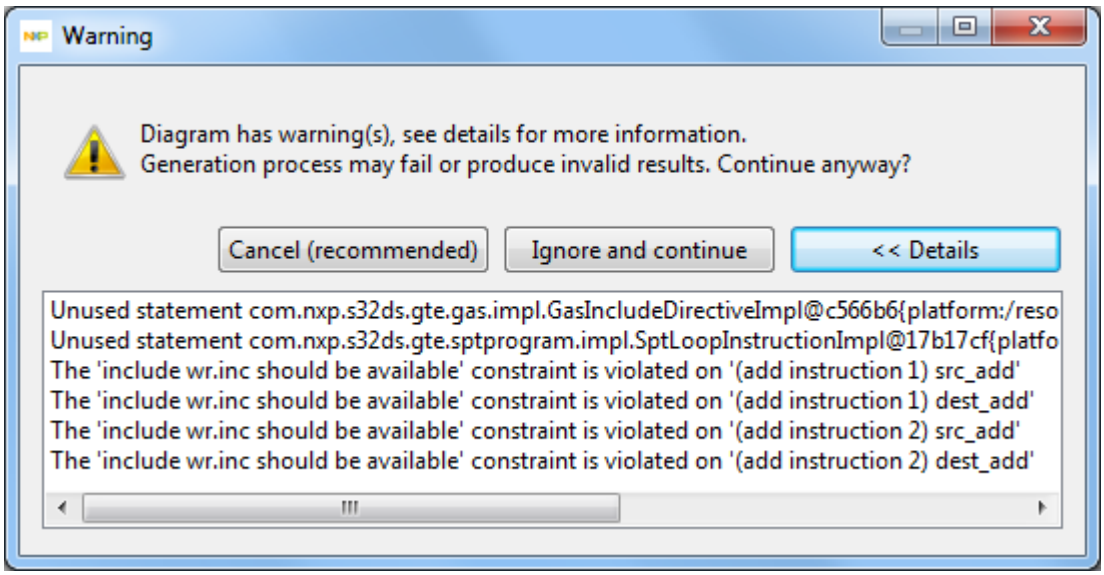
SPT generating

To generate the source code of SPT assembler you need to open the context menu of the diagram itself (by right-clicking on its background) and choose the **Generate SPT** action. The validation process starts. If validation finished successfully then:

- code generation starts. When code generation will be completed the message appears:

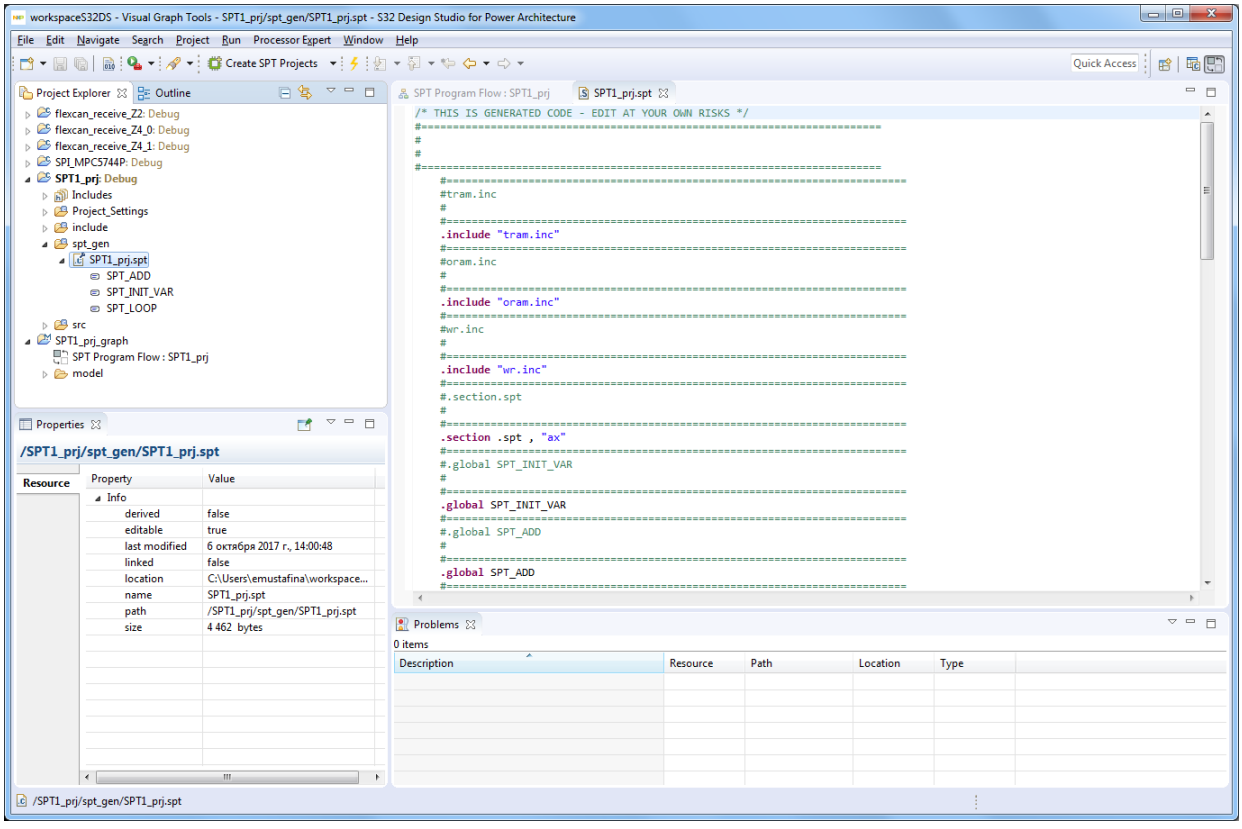


If the source code generation process fails, the following message is shown:



User can cancel code generation process by **Cancel** or ignore errors and finish the process by **Ignore and continue**.

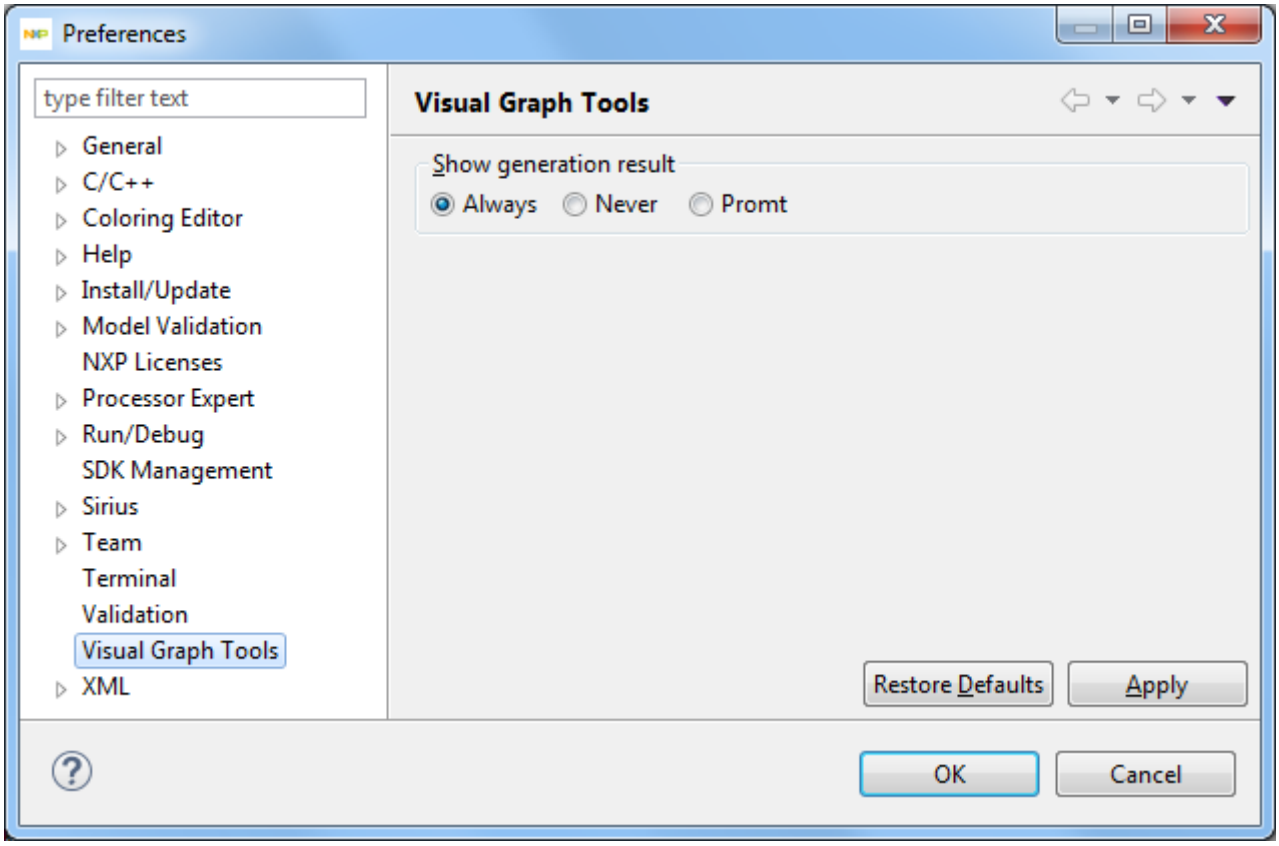
- the source code will be generated and displayed in the Editor view as **<Graph Tools project name>.spt** file



- the folder **spt_gen** with the **<Graph Tools project name>.spt** file will be created in the **Project Explorer** view
- the **Problems** view displays all errors and warnings.

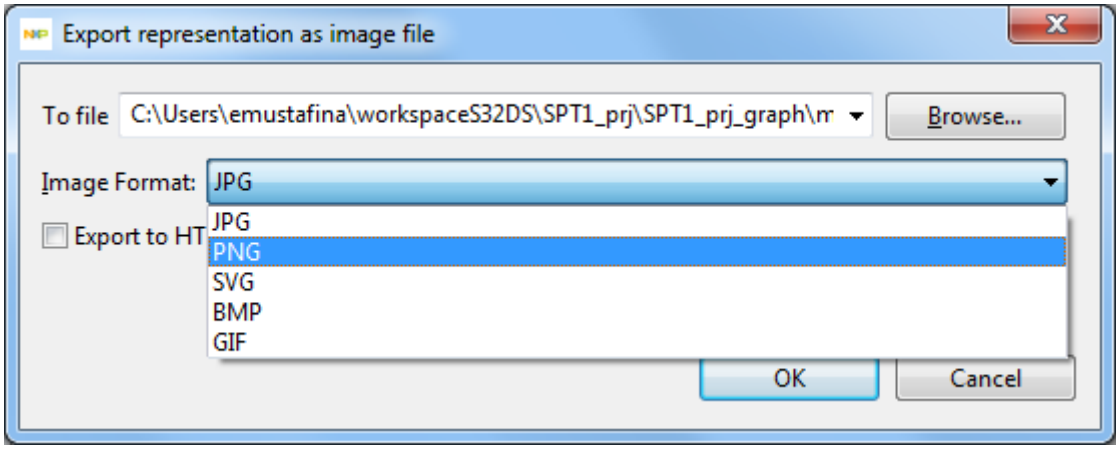
Code generation preferences

Displaying of code generation results can be set on the **Window** > **Preferences** > **Visual Graph Tools** pane:



Export diagram as image

To export diagram as image open the context menu of the diagram itself (by right-clicking on its background) and choose the **Export diagram as image** action. You can choose the destination folder and image format.

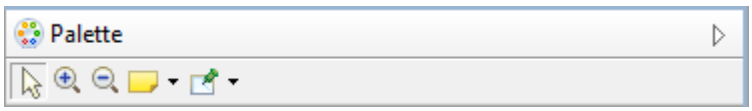


Palette

A diagram has the palette of tools, which by default is docked on the right hand side of the main graphical area. The top row of the palette contains some general tools which are available on the diagram.

Standard tools

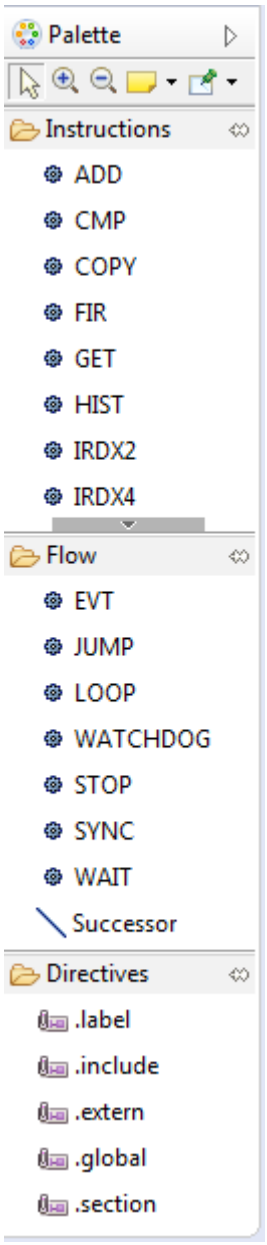
A few general tools are available on a diagram. They appear in the top row of the palette, just below the header.



Palette contents

Below the Palette's header, you will find all the tools which have been preconfigured for modeler – the next tool drawers:

- Instructions
- Flow
- Directives



Tool drawers

Tools in the palette are organized in expandable drawers to group them by category:

- **Instructions** (the instructions set depends on SPT version: SPT 1, SPT 2, or SPT 2.5):

Instructions set for SPT 1:

- ADD
- COPY
- FIR
- GET
- PDMA
- RDX2
- RDX4
- SET
- WIN

Instructions set for SPT 2 and SPT 2.5:

- ADD
- CMP
- COPY
- FIR
- GET
- HIST
- IRDX2
- IRDX4
- MAXS
- PDMA
- RDX2
- RDX4
- SCP
- SEL
- SET
- SUB
- VMT
- WIN

- **Flow** (the flows set depends on SPT version: SPT 1, SPT 2, or SPT 2.5):

Flows set for SPT 1:

- EVT
- LOOP
- STOP
- SYNC
- WAIT
- Successor

Flows set for SPT 2 and SPT 2.5:

- EVT
- JUMP
- LOOP
- WATCHDOG
- STOP
- SYNC
- WAIT
- Successor

- **Directives** (the directives set does not depend on SPT version):
  - .label
  - .include
  - .extern
  - .global
  - .section

To expand a drawer and show its content, simply click once in the drawer's header. Click again to fold it back and hide its tools. When you expand a drawer, some others may be folded automatically if there is not enough space to show all of them. To prevent this, you can «pin» a drawer opened by clicking on the «pin» icon in the drawer's title area when it is expanded. This will force it to be kept opened and its tools available, at all times.

Using Palette tools

There are different interaction modes possible for the tools available in the **Palette**. The basic pattern is to select the tool in the palette with a simple left-click on in it the palette, and then apply it once on the diagram.

- Direct Action Tools (all instruction tools and flow tools except the **Successor** tool). Direct action tools are the simplest and most common one. They require a single click somewhere on the diagram (as long as it is allowed). The most common direct action tools are element creation tools

- Edge Creation Tool (the **Successor** tool). Edge creation tools are a little more complex, in that they require the specification of both the source and target element of the new edge. Once the tool is selected, there are two possible usage modes:
  - First do a single click on the source element, then move the mouse to the target element (which may be the same) and click a second time to finish the edge creation
  - Alternatively, you can click on the source element, keep the mouse button pressed, move the mouse to the target element and release it there. Both methods are equivalent.

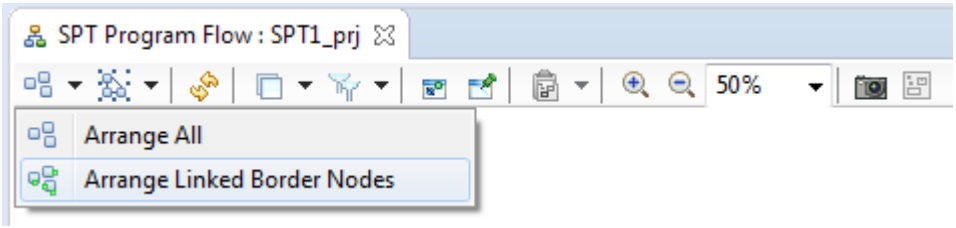
Normally, when you have used a tool once, it does not stay selected; if you want to reuse it a second time, you have to re-select it in the **Palette**. If you want to apply the same tool several times in a row, simply hold the **Ctrl** key pressed while using it; it will stay «armed» until you release the **Ctrl** key or select another tool.

To deselect a tool without executing it, simply press the **Esc** key, or select another one (for example the default Selection tool).

Toolbar

The top area of the diagram editors is filled with the toolbar, which provides access to many operations on diagrams and their elements. The content of the toolbar will depend on whether the current selection is the diagram itself (i.e. no element is selected) or one or several diagram elements.

When the diagram itself (and not a specific element) is selected, the toolbar contains the following buttons:

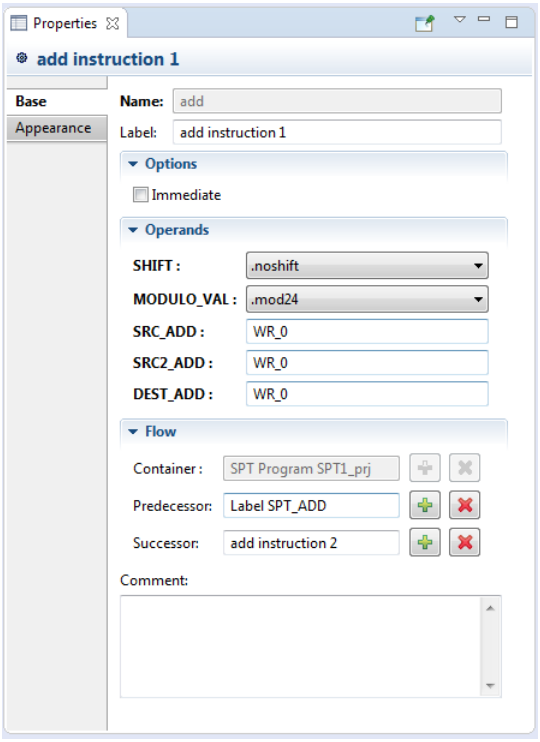


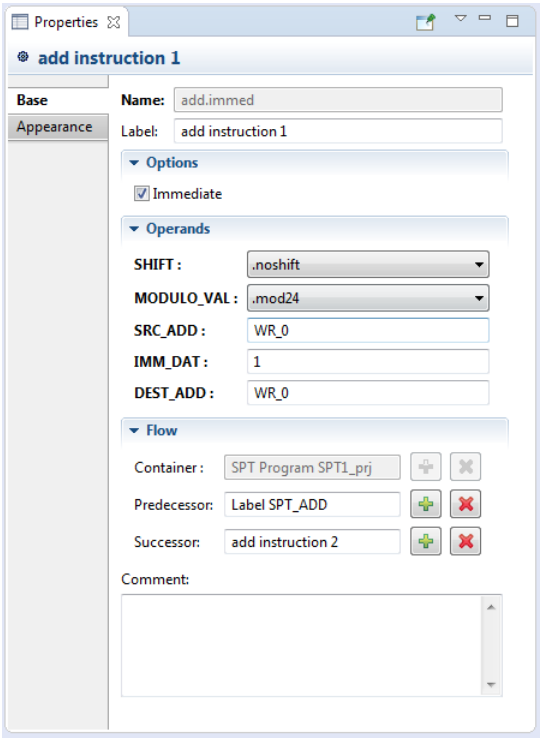
Properties view

In the **Properties** view, you will find several tabs which will not be the same if an element is selected on the diagram or not.

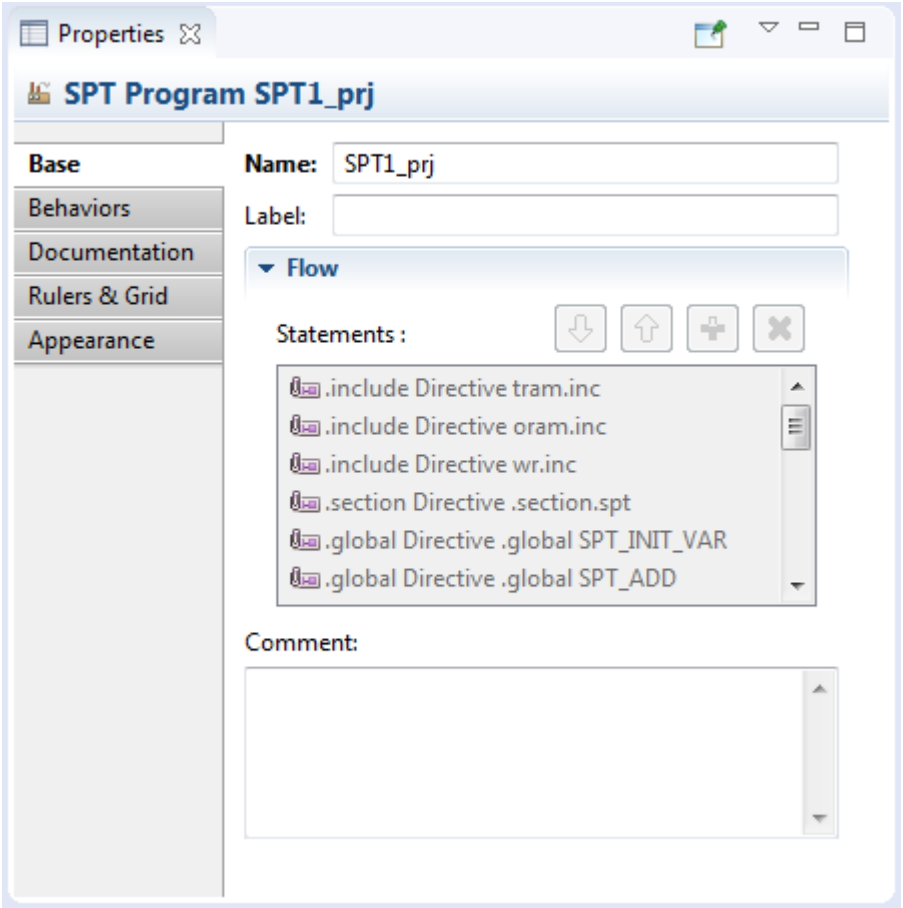
If an element is selected on the diagram, you will have on the **Properties** view the following tabs:

- **Appearance**: This tab allows to choose the style and size of text fields contained by the element
- **Base**: This tab allows managing a number of properties of selected diagram element. Names and values of parameters depend on the value of **Immediate** check box.





If you do not select an element, you will have in the **Properties** view the properties of the diagram itself:

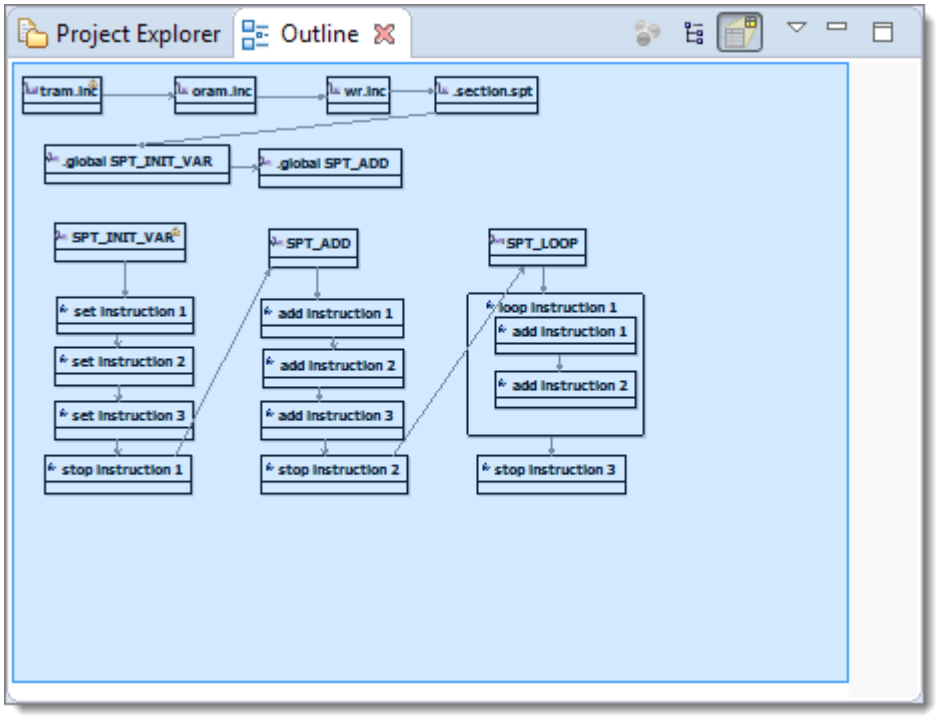


- **Behaviors** - this tab is not used (default Sirius feature)
- **Documentation** - this tab provides a text area for the current diagram documentation
- **Rulers & Grid** - allows the user to display vertical and horizontal rulers and also a grid on the diagram. You can choose how is displayed the grid (Solid style, Dash style, Dot style...)
- **Appearance** - this tab allows to choose the style and size of text fields of every elements of diagram which have not been specifically set on the Representation Diagram
- **Base** - this tab allows to manage properties of open diagram

Outline view

The **Outline** view has two display modes that you can select in its upper toolbar:

- In **Outline** mode, it shows the content of the model in a tree view
- In **Overview** mode, it shows a complete overview of the diagram. If the diagram is too big to be seen entirely on the diagram modeler, the overview is an easy way to navigate on the active diagram.



Preferences

Some preferences and configuration parameters are available for you to customize your experience. They can affect either the look or the behavior of diagrams. They are available through the standard preference UI (menu **Window** > **Preferences**), under the category **Sirius** > **Sirius Diagram** and its sub-categories.

How to Reach Us:

Home Page:

nxp.com

Web Support:

nxp.com/support

Revision October, 2017