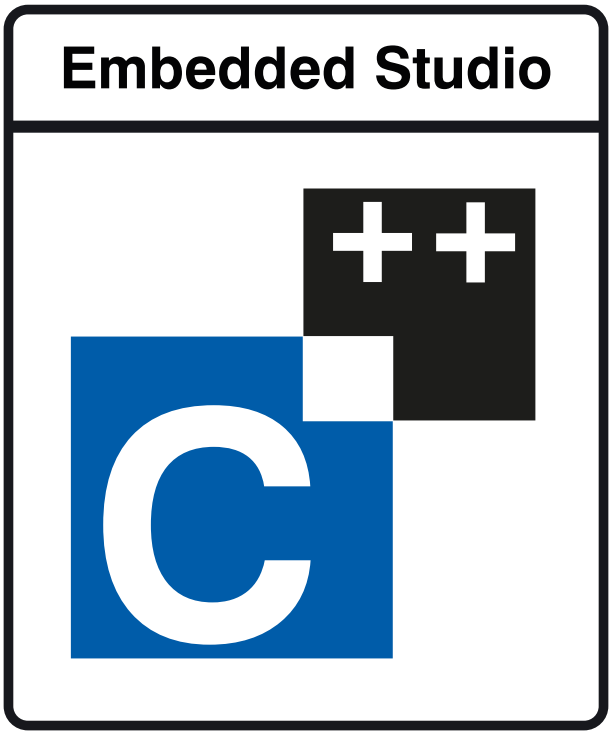
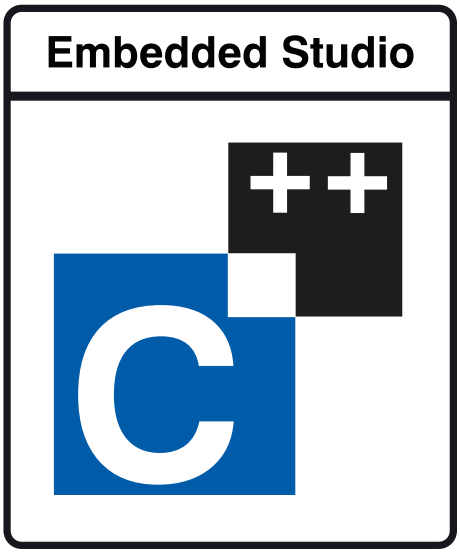
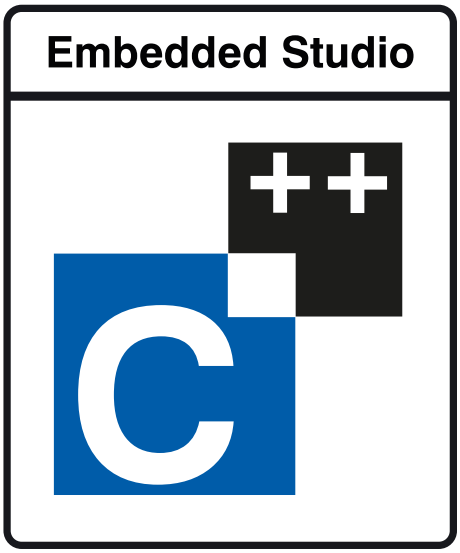


SEGGER Embedded Studio Reference Manual

Version: 8.26b



Contents



Introduction

This guide is divided into a number of sections:

Introduction

Covers installing SEGGER Embedded Studio on your machine and verifying that it operates correctly, followed by a brief guide to the operation of the SEGGER Embedded Studio integrated development environment, debugger, and other software supplied in the product.

SEGGER Embedded Studio User Guide

Contains information on how to use the SEGGER Embedded Studio development environment to manage your projects, build, and debug your applications.

hcc

Contains documentation for the C compiler, including syntax and usage details and a description of extensions provided by SEGGER Embedded Studio.

has

Contains detailed documentation covering how to use the assembler, the assembler notation, macros, and other assembler features.

What is SEGGER Embedded Studio?

C compiler

SEGGER Embedded Studio C is a faithful implementation of the ANSI and ISO standards for the programming language C. We have added some extensions that enhance usability in a microcontroller environment.

SEGGER Embedded Studio C Library

SEGGER Embedded Studio has its own royalty-free ANSI and ISO C compliant C library that has been specifically designed for use within embedded systems.

SEGGER Embedded Studio IDE

SEGGER Embedded Studio is a streamlined integrated development environment for building, testing, and deploying your applications. SEGGER Embedded Studio provides:

- *Source Code Editor*: A powerful source code editor with multi-level undo and redo, makes editing your code a breeze.
- *Project System*: A complete project system organizes your source code and build rules.
- *Build System*: With a single key press you can build all your applications in a solution, ready for them to be loaded onto a target microcontroller.
- *Debugger and Flash Programming*: You can download your programs directly into Flash and debug them seamlessly from within the IDE using a wide range of target interfaces.
- *Help system*: The built-in help system provides context-sensitive help and a complete reference to the SEGGER Embedded Studio IDE and tools.
- *Core Simulator*: As well as providing cross-compilation technology, SEGGER Embedded Studio provides a PC-based fully functional simulation of the target microcontroller core so you can debug parts of your application without waiting for hardware.

SEGGER Embedded Studio Tools

SEGGER Embedded Studio supplies command line tools that enable you to build your application on the command line using the same project file that the IDE uses.

What we don't tell you

This documentation does not attempt to teach the C or assembly language programming; rather, you should seek out one of the many introductory texts available. And similarly the documentation doesn't cover the SEGGER architecture or microcontroller application development in any great depth.

We also assume that you're fairly familiar with the operating system of the host computer being used.

C programming guides

These are must-have books for any C programmer:

- Kernighan, B.W. and Ritchie, D.M., *The C Programming Language* (2nd edition, 1988). Prentice-Hall, Englewood Cliffs, NJ, USA. ISBN 0-13-110362-8.
  The original C bible, updated to cover the essentials of ANSI C (1990 version).
- Harbison, S.P. and Steele, G.L., *C: A Reference Manual* (second edition, 1987). Prentice-Hall, Englewood Cliffs, NJ, USA. ISBN 0-13-109802-0.
  A nice reference guide to C, including a useful amount of information on ANSI C. Co-authored by Guy Steele, a noted language expert.

ANSI C reference

If you're serious about C programming, you may want to have the ISO standard on hand:

- ISO/IEC 9899:1990, C Standard and ISO/IEC 9899:1999, C Standard. The standard is available from your national standards body or directly from ISO at **http://www.iso.ch/**.

Getting Started

You will need to install a CPU support package:

- Choose **Tools** > **Package Manager**
- Choose the CPU support packages you wish to install and complete the dialog.

You will need to create a project:

- Choose **File** > **New Project**
- Select the appropriate Executable project type
- Specify a location for the project
- Complete the dialog selecting the appropriate **Target Processor** value

You will need to build the project:

- Choose **Build | Build 'Project'**

To debug on the simulator

- Choose **Project | Options...** to show the project options dialog
- In the **Search Options** type in **Simulator**
- Choose **Simulator** for the **Target Connection** option

To debug on hardware

- Choose **Project | Options...** to show the project options dialog
- In the **Search Options** type in **J-Link**
- Choose **J-Link** for the **Target Connection** option

To start debugging

- Choose **Debug | Go**

The debugger will stop the program at the main, you can now debug the application.

Text conventions

Menus and user interface elements

When this document refers to any user interface element, it will do so in **bold font**. For instance, you will often see reference to the **Project Explorer**, which is taken to mean the project explorer window. Similarly, you'll see references to the **Standard** toolbar which is positioned at the top of the SEGGER Embedded Studio window, just below the menu bar on Windows and Linux.

When you are directed to select an item from a menu in SEGGER Embedded Studio, we use the form *menu-name* > *item-name*. For instance, **File** > **Save** means that you need to click the **File** menu in the menu bar and then select the **Save** item. This form extends to items in sub-menus, so **File** > **Open With Binary Editor** has the obvious meaning.

Keyboard accelerators

Frequently-used commands are assigned keyboard *accelerators* to speed up common tasks. SEGGER Embedded Studio uses standard Windows and Mac OS keyboard accelerators wherever possible.

Windows and Linux have three key modifiers which are **Ctrl**, **Alt**, and **Shift**. For instance, **Ctrl+Alt+P** means that you should hold down the **Ctrl** and **Alt** buttons whilst pressing the **P** key; and **Shift+F5** means that you should hold down the **Shift** key whilst pressing **F5**.

Mac OS has four key modifiers which are # (command), # (option), # (control), and # (shift). Generally there is a one-to-one correspondence between the Windows modifiers and the Mac OS modifiers: **Ctrl** is #, **Alt** is #, and **Shift** is #. SEGGER Embedded Studio on Mac OS has its own set of unique key sequences using # (control) that have no direct Windows equivalent.

SEGGER Embedded Studio on Windows and Linux also uses *key chords* to expand the set of accelerators. Key chords are key sequences composed of two or more key presses. For instance, the key chord **Ctrl+T, D** means that you should type **Ctrl+T** followed by **D**; and **Ctrl+K, Ctrl+Z** means that you should type **Ctrl+T** followed by **Ctrl+Z**. Mac OS does not support accelerator key chords.

Code examples and human interaction

Throughout the documentation, text printed **in this typeface** represents verbatim communication with the computer: for example, pieces of C text, commands to the operating system, or responses from the computer. In examples, text printed *in this typeface* is not to be used verbatim: it represents a class of items, one of which should be used. For example, this is the format of one kind of compilation command:

hcl *source-file*

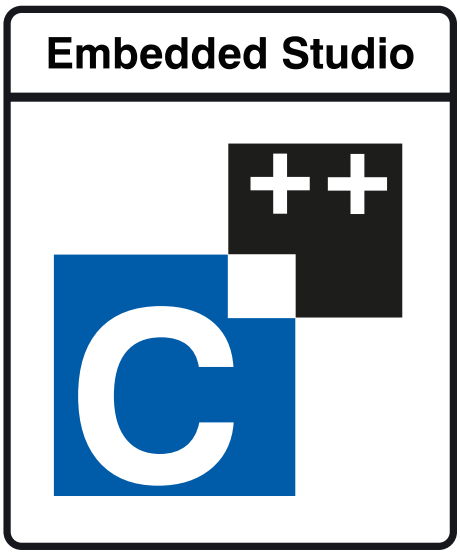
This means that the command consists of:

- The word **hcl**, typed exactly like that.
- A *source-file*: not the text **source-file**, but an item of the *source-file* class, for example **myprog.c**.

Whenever commands to and responses from the computer are mixed in the same example, the commands (i.e. the items which you enter) will be presented `in this typeface`. For example, here is a dialog with the computer using the format of the compilation command given above:

```
c:\code\examples>hcl -v myprog.c
```

The user types the text **hcl -v myprog.c** and then presses the enter key (which is assumed and is not shown); the computer responds with the rest.



SEGGER Embedded Studio User Guide

This is the user guide for the SEGGER Embedded Studio integrated development environment (IDE). The SEGGER Embedded Studio IDE consists of:

- a project system to organize your source files
- a build system to build your applications
- programmer aids to navigate and work effectively
- a target programmer to download applications into RAM or flash
- a debugger to pinpoint bugs

SEGGER Embedded Studio standard layout

SEGGER Embedded Studio's main window is divided into the following areas:

- *Title bar*: Displays the name of the current solution.
- *Menu bar*: Menus for editing, building, and debugging your program.
- *Toolbars*: Frequently used actions are quickly accessible on toolbars below the menu bar.
- *Editing area*: A tabbed view of any open editor windows and the HTML viewer.
- *Docked windows*: SEGGER Embedded Studio has many windows that dock to the left, right, or below the editing area. You can configure which windows will be visible, and their placement, when editing and debugging.
- *Status bar* At the bottom of the main window, the status bar contains useful information about the current editor, build status, and debugging environment.

Menu bar

The menu bar contains menus for editing, building, and debugging your program. You can navigate menus using the keyboard or the mouse.

Navigating menus using the mouse

To navigate menus using the mouse:

1. Click a menu title in the menu bar to show the related menu.
2. Click the desired command in the menu to execute that command.

—or—

1. Click and hold the mouse on a menu title in the menu bar to show the related menu.
2. Drag the mouse to the desired command in the menu.
3. Release the mouse while it is over the command to execute that command.

Navigating menus with the keyboard

To navigate menus using the keyboard:

1. Tap the **Alt** key activate the menu bar.
2. Tap **Return** to display the menu.
3. Use the **Left** and **Right** keys to select the required menu.
4. Use the **Up** or **Down** key to select the required command or submenu.
5. Press **Enter** to execute the selected command.
6. Press **Alt** or **Esc** at any time to cancel menu selection.

After you press the **Alt** key once, each menu on the menu bar has one letter underlined—its shortcut key. So, to activate a menu using the keyboard:

- While holding down the **Alt** key, type the desired menu's shortcut key.

After the menu appears, you can navigate it using the cursor keys:

- Use **Up** and **Down** to move up and down the list of menu items.
- Use **Esc** to cancel a menu.
- Use **Right** or **Enter** to open a submenu.
- Use **Left** or **Esc** to close a submenu and return to the parent menu.
- Type the underlined letter in a command's name to execute that command.

Title bar

The first item shown in the title bar is SEGGER Embedded Studio's name. Because SEGGER Embedded Studio can be used to target different processors, the name of the target processor family is also shown, to help you distinguish between instances of SEGGER Embedded Studio when debugging multi-processor or multi-core systems.

The filename of the active editor follows SEGGER Embedded Studio's name; you can configure the presentation of this filename as described below.

After the filename, the title bar displays status information on SEGGER Embedded Studio's state:

- **[building]** — SEGGER Embedded Studio is building a solution, building a project, or compiling a file.
- **[run]** — An application is running under control of SEGGER Embedded Studio's debugger.
- **[break]** — The debugger is stopped at a breakpoint.
- **[autostep]** — The debugger is single stepping the application without user interaction (*autostepping*).

Status bar

At the bottom of the window, the status bar contains useful information about the current editor, build status, and debugging environment. The status bar is divided into two regions: one contains a set of fixed panels and the other is used for messages.

The message area

The leftmost part of the status bar is a message area used for things such as status tips, progress information, warnings, errors, and other notifications.

Status bar panels

You can show or hide the following panels on the status bar:

Panel	Description
Target device status	Displays the connected target interface. When connected, this panel contains the selected target interface's name and, if applicable, the processor to which the target interface is connected. The LED icon flashes green when a program is running, is solid red when stopped at a breakpoint, and is yellow when connected to a target but not running a program. Double-clicking this panel displays the **Targets** pane, and right-clicking it invokes the **Target** shortcut menu.
Cycle count panel	Displays the number of processor cycles used by the executing program. This panel is only visible if the connected target supports performance counters that can report the total number of cycles executed. Double-clicking this panel resets the cycle counter to zero, and right-clicking it brings up the **Cycle Count** shortcut menu.
Insert/overwrite status	Indicates whether the current editor is in insert or overwrite mode. In overwrite mode, the panel displays "OVR"; in insert mode, the panel displays "INS".
Read-only status	Indicates whether the editor is in read-only mode. If the editor is editing a read-only file or is in read-only mode, the panel display "R/O"; if the editor is in read-write mode, the panel displays "R/W".
Build status	Indicates the success or failure of the last build. If the last build completed without errors or warnings, the build status pane contains **Built OK**; otherwise, it contains the number of errors and warnings reported. If there were errors, double-clicking this panel displays the **Build Log** in the **Output** pane.

Caret position	Indicates the insertion position position in the editor window. For text files, the **caret position** pane displays the line number and column number of the insertion point in the active window; when editing binary files, it displays the address being edited.
Time panel	Displays the current time.

Configuring the status bar panels

To configure which panels are shown on the status bar:

- Choose **View** > **Status Bar**.
- From the status bar menu, select the panels to display and deselect the ones you want hidden.

—or—

- Right-click the status bar.
- From the status bar menu, select the panels to display and deselect the ones you want to hide.

To show or hide the status bar:

- Choose **View** > **Status Bar**.
- From the status bar menu, select or deselect the **Status Bar** item.

You can choose to hide or display the *size grip* when SEGGER Embedded Studio's main window is not maximized. (The size grip is never shown in full-screen mode or when maximized.)

To show or hide the size grip

- Choose **View** > **Status Bar**.
- From the status bar menu, select or deselect the **Size Grip** item.

Editing workspace

The main area of SEGGER Embedded Studio is the editing workspace. It contains any files being edited, the on-line help system's HTML browser, and the Dashboard.

Docking windows

SEGGER Embedded Studio has a flexible docking system you can use to position windows as you like them. You can dock windows in the SEGGER Embedded Studio window or in the four *head-up display* windows. SEGGER Embedded Studio will remember the position of the windows when you leave the IDE and will restore them when you return.

Window groups

You can organize SEGGER Embedded Studio windows into *window groups*. A window group has multiple windows docked in it, only one of which is *active* at a time. The window group displays the active window's title for each of the windows docked in the group.

Clicking on the window icons in the window group's header changes the active window. Hovering over a docked window's icon in the header will display that window's title in a *tooltip*.

To dock a window to a different window group:

- Press and hold the left mouse button over the title of the window you wish to move.
- As you start dragging, all window groups, including hidden window groups, become visible.
- Drag the window over the window group to dock in.
- Release the mouse button.

Holding **Ctrl** when moving the window will prevent the window from being docked. If you do not dock a window on a window group, the window will float in a new window group.

Perspectives

SEGGER Embedded Studio remembers the dock position and visibility of each window in each *perspective*. The most common use for this is to lay your windows out in the **Standard** perspective, which is the perspective used when you are editing and not debugging. When SEGGER Embedded Studio starts to debug a program, it switches to the **Debug** perspective. You can now lay out your windows in this perspective and SEGGER Embedded Studio will remember how you laid them them out. When you stop debugging, SEGGER Embedded Studio will revert to the **Standard** perspective and that window layout for editing; when you return to **Debug** perspective on the next debug session, the windows will be restored to how you laid them out in that for debugging.

SEGGER Embedded Studio remembers the layout of windows, in all perspectives, such that they can be restored when you run SEGGER Embedded Studio again. However, you may wish to revert back to the standard docking positions; to do this:

Dashboard

When SEGGER Embedded Studio starts, it presents the **Dashboard**, a collection of panels that provide useful information, one-click loading of recent projects, and at-a-glance summaries of activity relevant to you.

Tasks

The **Tasks** panel indicates tasks you need to carry out before SEGGER Embedded Studio is fully functional—for instance, whether you need to activate SEGGER Embedded Studio, install packages, and so on.

Updates

The **Updates** panel indicates whether any packages you have installed are now out of date because a newer version is available. You can install each new package individually by clicking the **Install** button under each notification, or install all packages by clicking the **Install all updates** link at the bottom of the panel.

Projects

The **Projects** panel contains links to projects you have worked on recently. You can load a project by clicking the appropriate link, or clear the project history by clicking the **Clear List** button. To manage the contents of the list, click the **Manage Projects** link and edit the list of projects in the **Recent Projects** window.

News

The **News** panel summarizes the activity of any RSS and Atom feeds you have subscribed to. Clicking a link will display the published article in an external web browser. You can manage your feed subscriptions to by clicking the **Manage Feeds** link at the end of the **News** panel and *pinning* the feeds in the **Favorites** window—you are only subscribed to the pinned feeds.

Links

The **Links** panel is a handy set of links to your favorite websites. If you pin a link in the **Favorites** window, it appears in the **Links** panel.

SEGGER Embedded Studio help and assistance

SEGGER Embedded Studio provides context-sensitive help in increasing detail:

Tooltips

When you position the pointer over a button and keep it still, a small window displays a brief description of the button and its keyboard shortcut, if it has one.

Status tips

In addition to tooltips, SEGGER Embedded Studio provides a longer description in the status bar when you hover over a button or menu item.

Online manual

SEGGER Embedded Studio has links from all windows to the online help system.

The browser

Documentation pages are shown in the **Browser**.

Help using SEGGER Embedded Studio

SEGGER Embedded Studio provides an extensive, HTML-based help system that is available at all times.

To view the help text for a particular window or other user-interface element:

- Click to select the item with which you want assistance.
- Choose **Help** > **Help** or press **F1**.

Help within the text editor

The text editor is linked to the help system in a special way. If you place the insertion point within a word and press **F1**, the help-system page most likely to be useful is displayed in the HTML browser. This a great way to quickly find the help text for functions provided in the library.

Browsing the documentation

The **Contents** window lists all the topics in the SEGGER Embedded Studio documentation and gives a way to search through them.

The highlighted entry indicates the current help topic. When you click a topic, the corresponding page appears in the **Browser** window.

The **Next Topic** and **Previous Topic** items in the **Help** menu, or the buttons on the **Contents** window toolbar, help navigate through topics.

To search the online documentation, type a search phrase into the **Search** box on the **Contents** window toolbar.

To search the online documentation:

- Choose **Help** > **Contents** or press **Ctrl**+**Alt**+**F1**.
- Enter your search phrase in the **Search** box and press **Enter** (or **Return** on Macs).

The search commences and the table of contents is replaced by links to pages matching your query, listed in order of relevance. To clear the search and return to the table of contents, click the clear icon in the **Search** box.

Creating and managing projects

A SEGGER Embedded Studio *project* is a container for everything required to build your applications. It contains all the assorted resources and maintains the relationships between them.

A project is a convenient place to find every file and piece of information associated with your work. You place projects into a *solution*, which can contain one or more projects.

This chapter introduces the various parts of a project, shows how to create projects, and describes how to organize the contents of a project. It describes how to use the **Project Explorer** and **Project Manager** for project-management tasks.

Solutions and projects

To develop a product using SEGGER Embedded Studio, you must understand the concepts of *projects* and *solutions*.

- A *project* contains and organizes everything you need to create a single application or a library.
- A *solution* is a collection of projects and configurations.

Organizing your projects into a solution allows you to build all the projects in a solution with a single keystroke, and to load them onto the target ready for debugging.

In your SEGGER Embedded Studio project, you…

- …organize build-system inputs for building a product.
- …add information about items in the project, and their relationships, to assist you in the development process.

Projects in a solution can reside in the same or different directories. Project directories are always relative to the directory of the solution file, which enables you to more-easily move or share project-file hierarchies.

The **Project Explorer** organizes your projects and files, and provides quick access to the commands that operate on them. A toolbar at the top of the window offers quick access to commonly used commands.

Solutions

When you have created a solution, it is stored in a project file. Project files are text files, with the file extension **emProject**, that contain an XML description of your project. See **Project file format** for a description of the project-file format.

Projects

The projects you create within a solution have a *project type* SEGGER Embedded Studio uses to determine how to build the project. The project type is selected when you use the **New Project** dialog. The available project types depend on the SEGGER Embedded Studio variant you are using, but the following are present in most SEGGER Embedded Studio variants:

- *Executable*: — a program that can be loaded and executed.
- *Externally Built Executable*: — an executable that is not built by the SEGGER Embedded Studio internal build process.
- *Library*: — a group of object files collected into a single file (sometimes called an *archive*).
- *Externally Built Library*: — a library that is not built by the SEGGER Embedded Studio internal build process.

- *Object File*: — the result of a single compilation.
- *Staging*: — a project that will apply a user-defined command to each file in a project.
- *Combining*: — a project that can be used to apply a user-defined command when any files in a project have changed.

Project options and configurations

Project options are attached to project nodes. They are usually used in the build process, for example, to define C preprocessor symbols. You can assign different values to the same project option, based on a configuration: for example, you can assign one value to a C preprocessor symbol for release build and a different value for a debug build.

Folders and Dynamic Folders

Projects can contain *folders,* which are used to group related files. Automated grouping uses the files' extensions to, for example, put all .c files in one folder, etc. Grouping also can be done manually by explicitly creating a file within a folder. Note that these project folders do not map onto directories in the file system, they are used solely to structure the display of content shown in the **Project Explorer**.

Projects can also contain *dynamic folders* which will can show the directories and files contained in the file system in the project explorer. You can specify if the dynamic folder is recursive and use wildcards to include and exclude files.

Source files

Source files are all the files used to build a product. These include source code files and also section-placement files, memory-map files, and script files. All the source files you use for a particular product, or for a suite of related products, are managed in a SEGGER Embedded Studio project. A project can also contain files that are not directly used by SEGGER Embedded Studio to build a product but contain information you use during development, such as documentation. You edit source files during development using SEGGER Embedded Studio's built-in text editor, and you organize files into a target (described next) to define the build-system inputs for creating the product.

The source files of your project can be placed in folders or directly in the project. Ideally, the paths to files placed in a project should be relative to the project directory, but at times you might want to refer to a file in an absolute location and this is supported by the project system.

When you add a file to a project, the project system detects whether the file is in the project directory. If a file is not in the project directory, the project system tries to make a relative path from the file to the project directory. If the file isn't relative to the project directory, the project system detects whether the file is relative to

the **$(StudioDir)** directory; if so, the filename is defined using **$(StudioDir)**. If a file is not relative to the project directory or to **$(StudioDir)**, the full, absolute pathname is used.

The project system will allow (with a warning) duplicate files to be put into a project.

The project system uses a file's extension to determine the appropriate build action to perform on the file:

- A file with the extension **.c** will be compiled by a C compiler.
- A file with the extension **.s** or **.asm** will be compiled by an assembler.
- A file with the object-file extension **.hzo** will be linked.
- A file with the library-file extension **.hza** will be linked.
- A file with the extension **.xml** will be opened and its file type determined by the XML document type.
- Files with other file extensions will not be compiled or linked.

You can modify this behavior by setting a file's **File Type** project option with the **Common** configuration selected, which enables files with non-standard extensions to be compiled by the project system.

Externally Built Executables

You can use an external build process for **Externally Built Executable** project types by setting the **Build Command** project option, for example to **make target**. Alternatively you can set command lines for specific build steps to compile/assemble and link. When you create an **Externally Built Executable** project type configurations will be created that create command lines for a variety of external tool chains.

Solution links

You can create links to existing project files from a solution, which enables you to create hierarchical builds. For example, you could have a solution that builds a library together with a stub test driver executable. You can link to that solution from your current solution by right-clicking the solution node of the **Project Explorer** and selecting **Add Existing Project**. Your current solution can then use the library built by the other project.

Session files

When you exit SEGGER Embedded Studio, details of your current session are stored in a *session file*. Session files are text files, with the file extension **emSession**, that contain details such as which files you have opened in the editor and what breakpoints you have set in the **Breakpoint** window.

Creating a project

You can create a new solution for each project or place multiple projects in an existing solution.

To create a new project in an existing solution:

1. Choose **Project** > **Add New Project**.
2. In the **New Project** wizard, select the type of project you wish to create and specify where it will be placed.
3. Ensure that **Add the project to current solution** is checked.
4. Click **OK** to go to next stage or **Cancel** to cancel the project's creation.

The project name must be unique to the solution and, ideally, the project directory should be relative to the solution directory. The project system will use the project directory as the *current directory* when it builds your project. Once complete, the **Project Explorer** displays the new solution, project, and files contained in the project. To add another project to the solution, repeat the above steps.

To create a new project in a new solution:

1. Choose **File** > **New Project** or press **Ctrl**+**Shift**+**N**.
2. Select the type of project you wish to create and where it will be placed.
3. Click **OK**.

Adding existing files to a project

You can add existing files to a project in a number of ways.

To add existing files to the active project:

- Choose **Project** > **Add Existing File** or press **Ctrl**+**P, A**.

Using the **Open File** dialog, navigate to the directory containing the files and select the ones you wish to add to the project.

- Click **OK**.

The selected files are added to the folders whose filter matches the extension of each of the files. If no filter matches a file's extension, the file is placed underneath the project node.

To add existing files to a specific project:

1. In the **Project Explorer**, right-click the project to which you wish to add a new file.
2. Choose **Add Existing File**.

To add existing files to a specific folder:

1. In the **Project Explorer**, right-click the folder to which you wish to add a new file.
2. Choose **Add Existing File**.

The files are added to the specified folder without using filter matching.

To create a dynamic folder:

1. In the **Project Explore**, right click on the project to which you wish to add a new folder.
2. Choose **New Folder...**.
3. Using the **New Folder** dialog name the folder and then show the dynamic folder options.
4. Specify the required **Source Folder** and the **Filter Specification**.

The files that match the filter specification in the source folder will appear in the newly created folder.

Adding new files to a project

You can add new files to a project in a number of ways.

To add new files to the active project:

- Choose **Project** > **Add New File** or press **Ctrl**+**N**.

To add a new file to a project:

1. In the **Project Explorer**, right-click the project to which you wish to add a new file.
2. Choose **Add New File**.

When adding a new file, SEGGER Embedded Studio displays the **New File** dialog, from which you can choose the type of file to add, its filename, and where it will be stored. Once created, the new file is added to the folder whose filter matches the extension of the newly added file. If no filter matches the newly added file extension, the new file is placed underneath the project node.

To add new files to a folder:

1. In the **Project Explorer**, right-click the folder to which you wish to add a new file.
2. Choose **Add New File**.

The new file is added to the folder without using filter matching.

Removing a file, folder, project, or project link

You can remove whole projects, folders, or files from a project, or you can remove a project from a solution, using the **Remove** button on the **Project Explorer** toolbar. Note that removing a source file from a project does not remove it from disk.

To remove an item from the solution:

1. In the **Project Explorer**, select the item to remove.
2. Choose **Edit** > **Delete** or press **Del**.

—or—

1. In the **Project Explorer**, right-click the item to remove.
2. Choose **Remove**.

Building your application

SEGGER Embedded Studio builds your application using the resources and build rules it finds in your solution.

When SEGGER Embedded Studio builds your application, it tries to avoid building files that have not changed since they were last built. It does this by comparing the modification dates of the generated files with the modification dates of the dependent files together with the modification dates of the project options that pertain to the build. But if you are copying files, sometimes the modification dates may not be updated when the file is copied—in this instance, it is wise to use the **Rebuild** command rather than the **Build** command.

You can see the build rationale SEGGER Embedded Studio currently is using by setting the **Environment Options** > **Building** > **Show Build Information** environment option. To see the build commands themselves, set the **Environment Options** > **Building** > **Echo Build Command** environment option.

You may have a solution that contains several interdependent projects. Typically, you might have several executable projects and some library projects. The **Project Dependencies** dialog specifies the dependencies between projects and to see the effect of those dependencies on the solution build order. Note that dependencies can be set on a per-configuration basis, but the default is for dependencies to be defined in the **Common** configuration.

You will also notice that a new folder titled **Dependencies** has appeared in the **Project Explorer**. This folder contains the list of newly generated files and the files from which they were generated. To see if one of files can be decoded and displayed in the editor, right-click the file to see if the **View** command is available on the shortcut menu.

If you have the **Symbols** window open, it will be updated with the symbol and section information of all executable files built in the solution.

To generalize your builds, you can define macro values that are substituted when the project options are used. These macro values can be defined globally at the solution and project level, and can be defined on a per-configuration basis.

The combination of configurations, project options with inheritance, dependencies, and macros provides a very powerful build-management system. However, such systems can become complicated. To understand the implications of changing build settings, right-click a node in the **Project Explorer** and select **Options** to view a dialog that shows which macros and project options apply to that project node.

To build all projects in the solution:

1. Choose **Build** > **Build Solution** or press **Shift+F7**.

—or—

1. Right-click the solution in the **Project Explorer** window.
2. Choose **Build** from the shortcut menu.

To build a single project:

1. Select the required project in the **Project Explorer**.
2. Choose **Build** > **Build** or press **F7**.

—or—

1. Right-click the project in the **Project Explorer**.
2. Choose **Build**.

To compile a single file:

1. In the **Project Explorer**, click to select the source file to compile.
2. Choose **Build** > **Compile** or press **Ctrl+F7**.

—or—

1. In the **Project Explorer**, right-click the source file to compile.
2. Choose **Compile** from the shortcut menu.

Correcting errors after building

The results of a build are recorded in a **Build Log** that is displayed in the **Output** window. Errors are highlighted in red, warnings are highlighted in yellow. Double-clicking an error, warning, or note will move the insertion point to the line of source code that triggered that log entry.

You can move forward and backward through errors using **Search** > **Next Location** and **Search** > **Next Location**.

When you build a single project in a single configuration, the **Transcript** will display the memory used by the application and a summary for each memory area.

Creating variants using configurations

SEGGER Embedded Studio provides a facility to build projects in various configurations. Project configurations are used to create different software builds for your projects.

A configuration defines a set of project options. For example, the output of a compilation can be put into different directories, dependent upon the configuration. When you create a solution, some default project configurations are created.

Build configurations and their uses

Configurations are typically used to differentiate debug builds from release builds. For example, the compiler options for debug builds will differ from those of a release build: a debug build will set options so the project can be debugged easily, whereas a release build will enable optimization to reduce program size or to increase its speed. Configurations have other uses; for example, you can use configurations to produce variants of software, such as custom libraries for several different hardware variants.

Configurations inherit project options from other configurations. This provides a single point of change for definitions common to several configurations. A particular project option can be overridden in a particular configuration to provide configuration-specific settings.

When a solution is created, two configurations are generated — **Debug** and **Release** — and you can create additional configurations by choosing **Build** > **Build Configurations**. Before you build, ensure that the appropriate configuration is set using **Build** > **Set Active Build Configuration** or, alternatively, the **Active Configuration** combo box in the **Project Explorer**.

Selecting a configuration

To set the configuration that affects your building and debugging, use the combo box in the **Project Explorer** or select **Build** > **Set Active Build Configuration**

Creating a configuration

To create your own configurations, select **Build** > **Build Configurations** to invoke the **Configurations** dialog. The dialog has two sections, one to manage public configurations and one to manage private configurations.

Clicking the plus button in the public configurations section will allow you to create a new public configuration and then specify the private configurations to inherit.

Clicking the plus button in the private configuration section will allow you to create a new private configuration that can be inherited by public configurations.

Deleting a configuration

You can delete a configuration by selecting it and clicking the minus button. This deletion cannot be undone or canceled, so beware.

Project options

For solutions, projects, folders, and files, project options can be defined that are used by the project system in the build process. These project options can be viewed and modified by using the **Options** dialog in conjunction with the **Project Explorer**.

Some project options are only applicable to a given item type. For example, linker project options are only applicable to a project that builds an executable file. However, other project options can be applied either at the file, project, or solution project node. For example, a compiler project option can be applied to a solution, project, or individual file. By setting a project option at the solution level, you enable all files of the solution to use that project option's value.

Unique project options

A unique project option has *one* value. When a build is done, the value of a unique project option is the first one defined in the project hierarchy. For example, the **Treat Warnings As Errors** project option could be set to **Yes** at the solution level, which would then be applicable to every file in the solution that is compiled, assembled, and linked. You can then selectively define project options for other project items. For example, a particular source file may have warnings you decide are allowable, so you set the **Treat Warnings As Errors** to **No** for that particular file.

```
solution — Treat Warnings As Errors = Yes
    project1 — Treat Warnings As Errors = Yes
        file1 — Treat Warnings As Errors = Yes
        file2 — Treat Warnings As Errors = No
    project2 — Treat Warnings As Errors = No
        file1 — Treat Warnings As Errors = No
        file2 — Treat Warnings As Errors = Yes
```

In the above example, the files will be compiled with these values for **Treat Warnings As Errors**:

project1/file1	Yes
project1/file2	No
project2/file1	No
project2/file2	Yes

Aggregate project options

An aggregating project option collects all the values defined for it in the project hierarchy. For example, when a C file is compiled, the **Preprocessor Definitions** project option will take all the values defined at the file, project, and solution levels.

```
solution — Preprocessor Definitions = SolutionDef
```

```
    project1 – Preprocessor Definitions =
        file1 – Preprocessor Definitions =
        file2 – Preprocessor Definitions = File1Def
    project2 – Preprocessor Definitions = ProjectDef
        file1 – Preprocessor Definitions =
        file2 – Preprocessor Definitions = File2Def
```

In the above example, the files will be compiled with these preprocessor definitions:

project1/file1	SolutionDef
project1/file2	SolutionDef, File1Def
project2/file1	SolutionDef, ProjectDef
project2/file2	SolutionDef, ProjectDef, File2Def

Configurations and project options

Project options are defined for a configuration so you can have different values for a project option for different builds. A given configuration can inherit the project options of other configurations. When the project system requires a project option value, it checks for the existence of the project option value in the current configuration and then in the set of inherited configurations. You can specify the set of inherited configurations using the **Configurations** dialog.

A special configuration named **Common** is always inherited by a configuration. The **Common** configuration allows you to set project options that will apply to all configurations you create. If you are modifying a project option of your project, you almost certainly want each configuration to inherit it, so ensure that the **Common** configuration is selected.

If the project option is unique, the build system will use the one defined for the particular configuration. If the project option isn't defined for this configuration, the build system uses an arbitrary one from the set of inherited configurations.

If the option is still undefined, the build system uses the value for the **Common** configuration. If it is still undefined, the build system tries to find the value in the next higher level of the project hierarchy.

```
solution [Common] – Preprocessor Definitions = CommonSolutionDef

solution [Debug] – Preprocessor Definitions = DebugSolutionDef

solution [Release] – Preprocessor Definitions = ReleaseSolutionDef

    project1 - Preprocessor Definitions =

        file1 - Preprocessor Definitions =

        file2 [Common] – Preprocessor Definitions = CommonFile1Def

        file2 [Debug] – Preprocessor Definitions = DebugFile1Def

    project2 [Common] – Preprocessor Definitions = ProjectDef

        file1 – Preprocessor Definitions =

        file2 [Common] - Preprocessor Definitions = File2Def
```

In the above example, the files will be compiled with these preprocessor definitions when in **Debug** configuration…

File	Setting
project1/file1	CommonSolutionDef, DebugSolutionDef
project1/file2	CommonSolutionDef, DebugSolutionDef,CommonFile1Def, DebugFile1Def
project2/file1	CommonSolutionDef, DebugSolutionDef, ProjectDef

project2/file2	ComonSolutionDef, DebugSolutionDef, ProjectDef, File2Def

…and the files will be compiled with these **Preprocessor Definitions** when in **Release** configuration:

File	Setting
project1/file1	CommonSolutionDef, ReleaseSolutionDef
project1/file2	CommonSolutionDef, ReleaseSolutionDef, CommonFile1Def
project2/file1	CommonSolutionDef, ReleaseSolutionDef, ProjectDef
project2/file2	ComonSolutionDef, ReleaseSolutionDef, ProjectDef, File2Def

Project macros

You can use macros to modify the way the project system refers to files.

Macros are divided into four classes:

- *System macros* defined by SEGGER Embedded Studio relay information about the environment, such as paths to common directories.
- *Global macros* are saved in the environment and are shared across all solutions and projects. Typically, you would set up paths to libraries and any external items here.
- *Project macros* are saved as project options in the project file and can define values specific to the solution or project in which they are defined.
- *Build macros* are generated by the project system when you build your project.

System macros

System macros are defined by SEGGER Embedded Studio itself and as such are read-only. System macros can be used in project options, environment settings and to refer to files. See **System macros list** for the list of System macros.

Global macros

Global macros are store in the environment option **Build Macros**.

To define a global macro:

1. Use **Tools** > **Options** to show the environment options dialog.
2. In the **Environment Options** dialog's **Building** group, select the **Build Macros** option.
3. Click the ellipsis button on the right.
4. Set the macro using the syntax *name* = *replacement text*.

Project macros

To define a project macro:

To set the project macros:

1. Select the appropriate solution/project in the **Project Explorer**.
2. Use **Project** > **Options** to show the project options dialog.
3. In the **Project Options** dialog's **General Options** group, select the **Macros** option.
4. Click the ellipsis button on the right.
5. Set the macro using the syntax *name* = *replacement text*.

Build macros

Build macros are defined by the project system for a build of a given project node. See **Build macros list** for the list of build macros.

Using macros

You can use a macro for a project option or environment setting by using the $(macro) syntax. For example, the **Object File Name** option has a default value of `$(IntDir)/$(InputName)$(OBJ)`.

You can also specify a default value for a macro if it is undefined using the $(macro:default) syntax. For example, `$(MyMacro:0)` would expand to `0` if the macro `MyMacro` has not been defined.

Dependencies and build order

You can set up dependency relationships between projects using the **Project Dependencies** dialog. Project dependencies make it possible to build solutions in the correct order and, where the target permits, to load and delete applications and libraries in the correct order. A typical usage of project dependencies is to make an executable project dependent upon a library executable. When you elect to build the executable, the build system will ensure that the library it depends upon is up to date. In the case of a dependent library, the output file of the library build is supplied as an input to the executable build, so you don't have to worry about it.

Project dependencies are stored as project options and, as such, can be defined differently based upon the selected configuration. You almost always want project dependencies to be independent of the configuration, so the **Project Dependencies** dialog selects the **Common** configuration by default.

To make one project dependent upon another:

1. Choose **Project** > **Project Dependencies**.
2. From the **Project** dropdown, select the target project that depends upon other projects.
3. In the **Depends Upon** list box, select the projects the target project depends upon and deselect the projects it does not depend upon.

Some items in the **Depends Upon** list box may be dimmed, indicating that a circular dependency would result if any of those projects were selected. In this way, SEGGER Embedded Studio prevents you from constructing circular dependencies using the **Project Dependencies** dialog.

If your target supports loading multiple projects, the **Build Order** also reflects the order in which projects are loaded onto the target. Projects will load, in order, from top to bottom. Generally, libraries need to be loaded before the applications that use them, and you can ensure this happens by making the application dependent upon the library. With this dependency set, the library gets built and loaded before the application does.

Applications are deleted from a target in reverse of their build order; in this way, applications are removed before the libraries on which they depend.

Precompile Header File support

You can specify a single file in your project to be a precompiled header by setting the project option **Precompiled Header File** on the file node of the project. The file should be project local i.e. in the same directory as the project file and should include the header files that you wish to be compiled.

You must set the project level option **Enable Precompiled Header File** which supplies the output file generated by the precompiled header file to the compilation of each file in the project.

Linking and section placement

Executable programs consist of a number of sections. Typically, there are program sections for code, initialized data, and zeroed data. There is often more than one code section and they must be placed at specific addresses in memory.

To describe how the program sections of your program are positioned in memory, the SEGGER Embedded Studio project system uses *memory-map* files and *section-placement* files. These XML-formatted files are described in **Memory Map file format** and **Section Placement file format**. They can be edited with the SEGGER Embedded Studio text editor. The memory-map file specifies the start address and size of target memory segments. The section-placement file specifies where to place program sections in the target's memory segments. Separating the memory map from the section-placement scheme enables a single hardware description to be shared across projects and also enables a project to be built for a variety of hardware descriptions.

For example, a memory-map file representing a device with two memory segments called **FLASH** and **SRAM** could look something like this in the memory-map editor.

```
<Root name="Device1">
  <MemorySegment name="FLASH" start="0x10000000" size="0x10000" />
  <MemorySegment name="SRAM" start="0x20000000" size="0x1000" />
```

A corresponding section-placement file will refer to the memory segments of the memory-map file and will list the sections to be placed in those segments. This is done by using a memory-segment name in the section-placement file that matches the corresponding memory-segment name in the memory-map file.

For example, a section-placement file that places a section called **.stack** in the **SRAM** segment and the **.vectors** and **.text** sections in the **FLASH** segment would look like this:

```
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH" >
    <ProgramSection name=".vectors" load="Yes" />
    <ProgramSection name=".text" load="Yes" />
  </MemorySegment>
  <MemorySegment name="SRAM" >
   <ProgramSection name=".stack" load="No" />
  </MemorySegment>
</Root>
```

Note that the order of section placement within a segment is top down; in this example **.vectors** is placed at lower addresses than **.text**.

The memory-map file and section-placement file to use for linkage can be included as a part of the project or, alternatively, they can be specified in the project's **linker options**.

You can create a new program section using either the assembler or the compiler. For the C compiler, this can be achieved using one of the **#pragma** directives. For example:

```
#pragma codeseg(".foo")
```

```
void foobar(void);
#pragma codeseq(default)
```

This will allocate **foobar** in the section called **.foo**. Alternatively, you can specify the names for the code, constant, data, and zeroed-data sections of an entire compilation unit by using the **Section Options** options.

You can now place the section into the section placement file using the editor so that it will be located after the vectors sections as follows:

```
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH">
    <ProgramSection name=".vectors" load="Yes" />
    <ProgramSection name=".foo" load="Yes" />
    <ProgramSection name=".text" load="Yes" />
  </MemorySegment>
  <MemorySegment name="SRAM">
   <ProgramSection name=".stack" load="No" />
  </MemorySegment>
</Root>
```

If you are modifying a section-placement file that is supplied in the SEGGER Embedded Studio distribution, you will need to import it into your project using the **Project Explorer**.

Sections containing code and constant data should have their **load** project option set to **Yes**. Some sections don't require any loading, such as stack sections and zeroed-data sections; such sections should have their **load** project option set to **No**.

You can specify that initialization data is stored in the default program section using the `INIT` directive, and you can refer to the start and end of the section using the SFE and SFB directives. If, for example, you create a new data section called `IDATA2`, you can store this in the program by putting the following into the startup code:

```
_data2_init_begin::
  INIT "IDATA2"
_data2_init_end::
```

You can then use these symbols to copy the stored section information into the data section using (an assembly-coded version of):

```
/* Section image located in flash */
extern const unsigned char data2_init_begin[];
extern const unsigned char data2_init_end[];

memcpy(SFB(IDATA2), data2_init_begin, data2_init_end-data2_init_end)
```

Using source control

Source control is an essential tool for individuals or development teams. SEGGER Embedded Studio integrates with several popular source-control systems to provide this feature for files in your SEGGER Embedded Studio projects.

Source-control capability is implemented by a number of third-party providers, but the set of functions provided by SEGGER Embedded Studio aims to be provider independent.

Source control capabilities

The source-control integration capability provides:

- Connecting to the source-control *repository* and mapping files in the SEGGER Embedded Studio project to those in source control.
- Showing the source-control status of files in the project.
- Adding files in the project to source control.
- Fetching files in the project from source control.
- Optionally locking and unlocking files in the project for editing.
- Comparing a file in the project with the latest version in source control.
- Updating a file in the project by merging changes from the latest version in source control.
- Committing changes made to project files into source control.

Configuring source-control providers

SEGGER Embedded Studio supports Subversion, Git, and Mercurial as source-control systems. To enable SEGGER Embedded Studio to utilize source-control features, you need to install, on your operating system, the appropriate command line client for the source-control systems that you will use.

Once you have installed the command line client, you must configure SEGGER Embedded Studio to use it.

To configure Subversion:

1. Choose **Tools** > **Options** or press **Alt+,**.
2. Select the **Source Control** category in the options dialog.
3. Set the **Executable** environment option of the **Subversion Options** group to point to Subversion `svn` command. On Windows operating systems, the Subversion command is `svn.exe`.

To configure Git:

1. Choose **Tools** > **Options** or press **Alt+,**.
2. Select the **Source Control** category in the options dialog.
3. Set the **Executable** environment option of the **Git Options** group to point to Git `git` command. On Windows operating systems, the Git command is `git.exe`.

To configure Mercurial:

1. Choose **Tools** > **Options** or press **Alt+,**.
2. Select the **Source Control** category in the options dialog.
3. Set the **Executable** environment option of the **Mercurial Options** group to point to Git `hg` command. On Windows operating systems, the Git command is `hg.exe`.

Connecting to the source-control system

When SEGGER Embedded Studio loads a project, it examines the file system folder that contains the project to determine the source-control system the project uses. If SEGGER Embedded Studio cannot determine, from the file system, the source-control system in use, it disables source-control integration.

That is, if you have not set up the paths to the source-control command line clients, even if a working copy exists and the appropriate command line client is installed, SEGGER Embedded Studio cannot establish source-control integration for the project.

User credentials

You can set the credentials that the source-control system uses, for commands that require credentials, using **VCS** > **Options** > **Configure**. From here you can set the user name and password. These details are saved to the session file (the password is encrypted) so you won't need to specify this information each time the project is loaded.

Note

SEGGER Embedded Studio has no facility to create repositories from scratch, nor to clone, pull, or checkout repositories to a working copy: it is your responsibility to create a working copy outside of SEGGER Embedded Studio using your selected command-line client or Windows Explorer extension.

The "Tortoise" products are a popular set of tools to provide source-control facilities in the Windows shell. Use Google to find **TortoiseSVN**, **TortoiseGit**, and **TortoiseHG** and see if you like them.

File source-control status

Determining the source-control status of a file can be expensive for large repositories, so SEGGER Embedded Studio updates the source-control status in the background. Priority is given to items that are displayed.

A file will be in one of the following states:

- *Clean*: The file is in source control and matches the tip revision.
- *Not Controlled*: The file is not in source control.
- *Conflicted*: The file is in conflict with changes made to the repository.
- *Locked*: The file is locked.
- *Update Available*: The file is older than the most-recent version in source control.
- *Added*: The file is scheduled to be added to the repository.
- *Removed*: The file is scheduled to be removed from the repository.

If the file has been modified, its status is displayed in red in the **Project Explorer**. Note that if a file is not under the local root, it will not have a source-control status.

You can reset any stored source-control file status by choosing **VCS** > **Refresh**.

Source-control operations

Source-control operations can be performed on single files or recursively on multiple files in the **Project Explorer** hierarchy. Single-file operations are available on the **Source Control** toolbar and on the text editor's shortcut menu. All operations are available using the **VCS** menu. The operations are described in terms of the **Project Explorer** shortcut menu.

Adding files to source control

To add files to the source-control system:

1. In the **Project Explorer**, select the file to add. If you select a folder, project, or solution, any eligible child items will also be added to source control.
2. choose **Source Control** > **Add** or press **Ctrl+R, A**.
3. The dialog will list the files that can be added.
4. In that dialog, you can deselect any files you don't want to add to source control.
5. Click **Add**.

Note

Files are scheduled to be added to source control and will only be committed to source control (and seen by others) when you commit the file.

Enabling the **VCS** > **Options** > **Add Immediately** option will bypass the dialog and immediately add (but not commit) the files.

Updating files

To update files from source control:

1. In the **Project Explorer**, select the file to update. If you select a folder, project, or solution, any eligible child items will also be updated from source control.
2. choose **Source Control** > **Update** or press **Ctrl+R, U**.
3. The dialog will list the files that can be updated.
4. In that dialog, you can deselect any files you don't want to update from source control.
5. Click **Update**.

Note

Enabling the **VCS** > **Options** > **Update Immediately** option will bypass the dialog and immediately update the files.

Committing files

To commit files:

1. In the **Project Explorer**, select the file to commit. If you select a folder, project, or solution, any eligible child items will also be committed.
2. Choose **Source Control** > **Commit** or press **Ctrl+R, C**.
3. The dialog will list the files that can be committed.
4. In that dialog, you can deselect any files you don't want to commit and enter an optional comment.
5. Click **Commit**.

Note

Enabling the **VCS** > **Options** > **Commit Immediately** option will bypass the dialog and immediately commit the files without a comment.

Reverting files

To revert files:

1. In the **Project Explorer**, select the file to revert. If you select a folder, project, or solution, any eligible child items will also be reverted.
2. Choose **Source Control** > **Revert** or press **Ctrl+R, V**.
3. The dialog will list the files that can be reverted.
4. In that dialog, you can deselect any files you don't want to revert.
5. Click **Revert**.

Note

Enabling the **VCS** > **Options** > **Revert Immediately** option will bypass the dialog and immediately revert files.

Locking files

To lock files:

1. In the **Project Explorer**, select the file to lock. If you select a folder, project, or solution, any eligible child items will also be locked.
2. Choose **Source Control** > **Lock** or press **Ctrl+R, L**.
3. The dialog will list the files that can be locked.
4. In that dialog, you can deselect any files you don't want to lock and enter an optional comment.
5. Click **Lock**.

Note

Enabling the **VCS** > **Options** > **Lock Immediately** option will bypass the dialog and immediately lock files without a comment.

Unlocking files

To unlock files:

1. In the **Project Explorer**, select the file to lock. If you select a folder, project, or solution, any eligible child items will also be unlocked.
2. Choose **Source Control** > **Unlock** or press **Ctrl+R, N**.
3. The dialog will list the files that can be unlocked.
4. In that dialog, you can deselect any files you don't want to unlock.
5. Click **Unlock**.

Note

Enabling the **VCS** > **Options** > **Unlock Immediately** option will bypass the dialog and immediately unlock files.

Removing files from source control

To remove files from source control:

1. In the **Project Explorer**, select the file to remove. If you select a folder, project, or solution, any eligible child items will also be removed.
2. choose **Source Control** > **Remove** or press **Ctrl+R, R**.
3. The dialog will list the files that can be removed.
4. In that dialog, you can deselect any files you don't want to remove.
5. Click **Remove**.

Note

Files are scheduled to be removed from source control and will still be and seen by others, giving you the opportunity to revert the removal. When you commit the file, the file is removed from source control.

Enabling the **VCS** > **Options** > **Remove Immediately** option will bypass the dialog and immediately remove (but not commit) files.

Showing differences between files

To show the differences between the file in the project and the version checked into source control, do the following:

1. In the **Project Explorer**, right-click the file.
2. From the shortcut menu, choose **Source Control** > **Show Differences**.

You can use an external diff tool in preference to the built-in SEGGER Embedded Studio diff tool. To define the diff command line SEGGER Embedded Studio generates, choose **Tools** > **Options** > **Source Control** > **Diff Command Line**. The command line is defined as a list of strings to avoid problems with spaces in arguments. The diff command line can contain the following macros:

- *$(localfile)*: The filename of the file in the project.
- *$(remotefile)*: The filename of the latest version of the file in source control.
- *$(localname)*: A display name for $(localfile).
- *$(remotename)*: A display name for $(remotefile).

Source-control properties

When a file in the project is in source control, the **Properties** window shows the following properties in the **Source Control Options** group:

Property	Description
SEGGER Embedded Studio Status	The source-control status of working copy as viewed by SEGGER Embedded Studio.
last Author	The author of the file's head revision.
Path: Relative	The item's path relative to the repository root.
Path: Repository	The pathname of the file in the source-control system, typically a URL.
Path: Working Copy	The pathname of the file in the working copy.
Provider	The name of the source-control system managing this file.
Provider Status	The status of the file as reported by the source-control provider.
Revision: Local	The revision number/name of the local file.
Revision: Remote	The revision number/name of the most-recent version in source control.
Status: In Conflict?	If **Yes**, updates merged into the file using **Update** conflict with the changes you made locally; if **No**, the file is not locked. When conflicted, must resolve the conflicts and mark them `Resolved` before committing the file.
Status: Locked?	If **Yes**, the file is lock by you; if **No**, the file is not locked.
Status: Modified?	If **Yes**, the checked-out file differs from the version in the source control system; if **No**, they are identical.
Status: Update Available?	If **Yes**, the file in the project location is an old version compared to the latest version in the source-control system—use **Update** to merge in the latest changes.

Subversion provider

The Subversion source-control provider has been tested with SVN 1.4.3.

Provider-specific options

The following environment options are supported:

Property	Description
Executable	The path to the `svn` executable.
Lock Supported	If **Yes**, check out and undo check out operations are supported. Check out will issue the `svn lock` command; check in and undo check out will issue the `svn unlock` command.
Authentication	Selects whether authentication (user name and password) is sent with every command.
Show Updates	Selects whether the update (`-u` flag) is sent with status requests in order to show that new versions are available in the repository. Note that this requires a live connection to the repository: if you are working without a network connection to your repository, you can disable this switch and continue to enjoy source control status information in the **Project Explorer** and **Pending Changes** windows.

Connecting to the source-control system

When connecting to source control, the provider checks if the local root is in SVN control. If this is the case, the local and remote root will be set accordingly. If the local root is not in SVN control after you have set the remote root, a `svn checkout -N` command will be issued to make the local root SVN controlled. This command will also copy any files in the remote root to the local root.

The user name and password you enter will be supplied with each svn command the provider issues.

Source control operations

The SEGGER Embedded Studio source-control operations are implemented using Subversion commands. Mapping SEGGER Embedded Studio source-control operations to Subversion source-control operations is straightforward:

Operation	Command
Commit	`svn commit` for the file, with optional comment.
Update	`svn update` for each file.

Revert	`svn revert` for each file.
Resolved	`svn resolved` for each file.
Lock	`svn lock` for each file, with optional comment.
Unlock	`svn unlock` for each file.
Add	`svn add` for each file.
Remove	`svn remove` for each file.
Source Control Explorer	`svn list` with a remote root. `svn mkdir` to create directories in the repository.

CVS provider

The CVS source-control provider has been tested with CVSNT 2.5.03. The CVS source-control provider uses the CVS `rls` command to browse the repository—this command is implemented in CVS 1.12 but usage of '.' as the root of the module name is not supported.

Provider-specific options

The following environment options are supported:

Property	Description
CVSROOT	The CVSROOT value to access the repository.
Edit/Unedit Supported	If **Yes**, **Check Out** and **Undo Check Out** commands are supported. Any check-out operation will issue the `cvs edit` command; any check-in or undo-check-out operation will issue the `cvs unedit` command; the status operation will issue the `cvs ss` command.
Executable	The path to the `cvs` executable.
Login/Logout Required	If **Yes**, **Connect** will issue the `cvs login` command.

Connecting to the source-control system

When connecting to source control, the provider checks if the local root is in CVS control. If this is the case, the local and remote root will be set accordingly. If the local root is not in CVS control after you have set the remote root, a `cvs checkout -l -d` command will be issued to make the local root CVS controlled. This command will also copy any files in the remote root to the local root.

Source-control operations

The SEGGER Embedded Studio source-control operations have been implemented using CVS commands. There are no multiple-file operations, each operation is done on a single file and committed as part of the operation.

Operation	Command
Get Status	`cvs status` and optional `cvs editors` for local directories in CVS control. `cvs rls -e` for directories in the repository.
Add To Source Control	`cvs add` for each directory not in CVS control. `cvs add` for the file. `cvs commit` for the file and directories.
Get Latest	`cvs update -l -d` for each directory not in CVS control. `cvs update` to merge the local file. `cvs update -C` to overwrite the local file.

Check Out	Optional `cvs update -C` to get the latest version. `cvs edit` to lock the file.
Undo Check Out	`cvs unedit` to unlock the file. Optional `cvs update` to get the latest version.
Check In	`cvs commit` for the file.
Source Control Explorer	`cvs rls -e` with a remote root starting with '.'. `cvs import` to create directories in the repository.

Package management

Additional target-support functions can be added to, and removed from, SEGGER Embedded Studio with *packages*.

A SEGGER Embedded Studio package is an archive file containing a collection of target-support files. Installing a package involves copying the files it contains to an appropriate destination directory and registering the package with SEGGER Embedded Studio's package system. Keeping target-support files separate from the main SEGGER Embedded Studio installation allows us to support new hardware and issue bug fixes for existing hardware-support files between SEGGER Embedded Studio releases, and it allows third parties to develop their own support packages.

Installing packages

Use the **Package Manager** to automate the download, installation, upgrade and removal of packages.

To activate the Package Manager:

- Choose **Tools** > **Manage Packages**.

In some situations, such as using SEGGER Embedded Studio on a computer without Internet access or when you want to install packages that are not on the website, you cannot use the **Package Manager** to install packages and it will be necessary to manually install them.

To manually install a package:

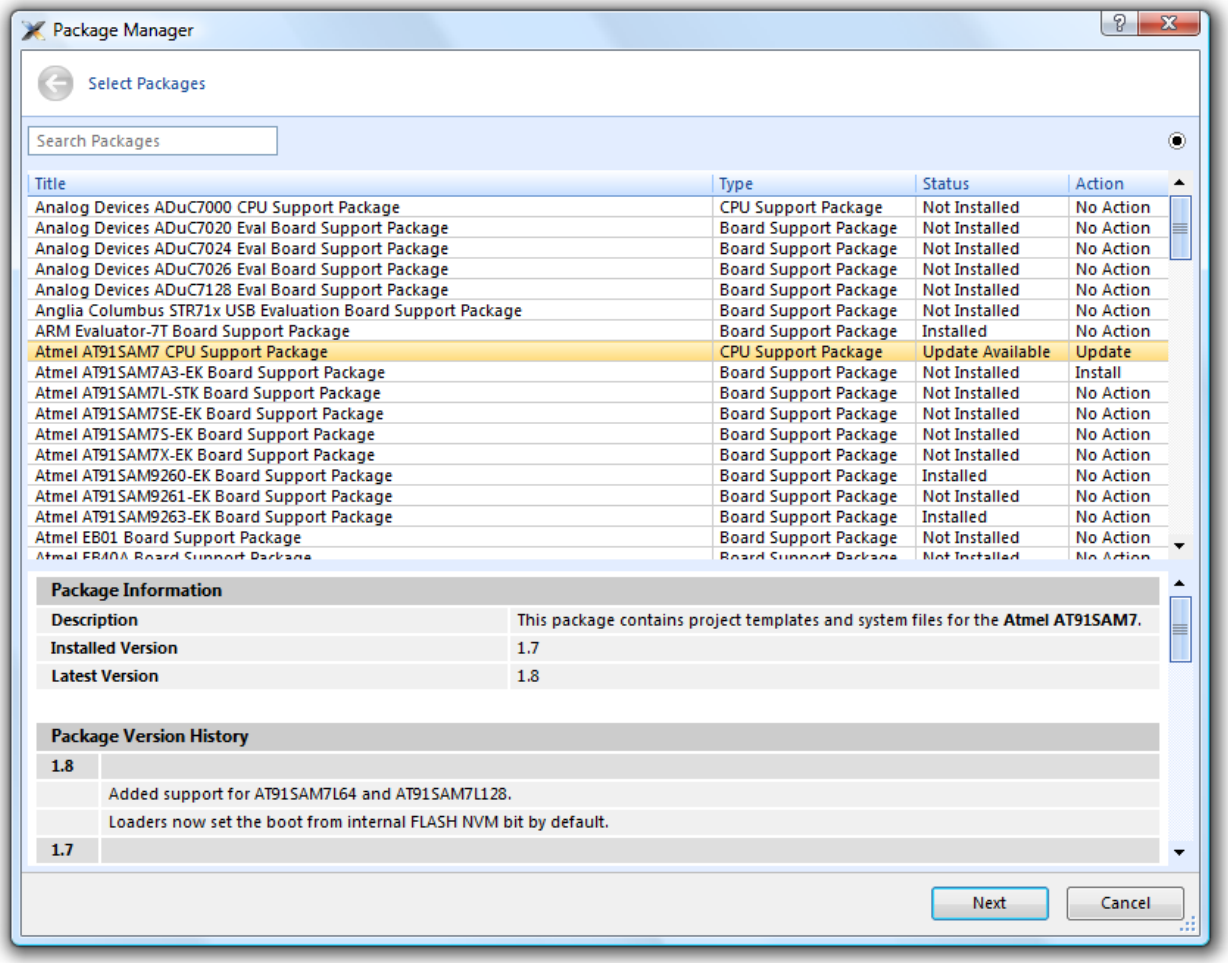
1. Choose **Tools** > **Manually Install Packages**.
2. Select one or more package files you want to install.
3. Click **Open** to install the packages.

Choose **Tools** > **Show Installed Packages** to see more information on the installed packages.

The **Package Manager** window will remove manually installed packages.

The package manager

The **Package Manager** manages the support packages installed on your system. It lists the available packages, shows the installed packages, and allows you to install, update, reinstall, and remove them.



To activate the Package Manager:

- Choose **Tools** > **Manage Packages**.

Filtering the package list

By default, the **Package Manager** lists all available and installed packages. You can filter the displayed packages in a number of ways.

To filter by package status:

- Click on the disclosure icon near the top-right corner of the dialog.
- Use the pop-up menu to choose how to filter the list of packages.

The list-filter choices are:

- **Display All** — Show all packages irrespective of their status.
- **Display Not Installed** — Show packages that are available but are not currently installed.

- **Display Installed** — Only show packages that are installed.
- **Display Updates** — Only show packages that are installed but are not up-to-date because a newer version is available.

You can also filter the list of packages by the text in the package's title and documentation.

To filter packages by keyword:

- Type the keyword into the **Search Packages** box at the top-left corner of the dialog.

Installing a package

The package-installation operation downloads a package to **$(PackagesDir)/downloads**, if it has not been downloaded already, and unpacks the files contained within the package to their destination directory.

To install a package:

1. Choose **Tools** > **Package Manager** and set the status filter to **Display Not Installed**.
2. Select the package or packages you wish to install.
3. Right-click the selected packages and choose **Install Selected Packages** from the shortcut menu.
4. Click **Next**; you will be see the actions the **Package Manager** is about to carry out.
5. Click **Next** and the **Package Manager** will install the selected packages.
6. When installation is complete, click **Finish** to close the **Package Manager**.

Updating a package

The package-update operation first removes existing package files, then it downloads the updated package to **$(PackagesDir)/downloads** and unpacks the files contained within the package to their destination directory.

To update a package:

1. Choose **Tools** > **Package Manager** and set the status filter to **Display Updates**.
2. Select the package or packages you wish to update.
3. Right-click the selected packages and choose **Update Selected Packages** from the shortcut menu.
4. Click **Next**; you will see the actions the **Package Manager** is about to carry out.
5. Click **Next** and the **Package Manager** will update the package(s).
6. When the update is complete, click **Finish** to close the **Package Manager**.

Removing a package

The package-remove operation removes all the files that were extracted when the package was installed.

To remove a package:

1. Choose **Tools** > **Package Manager** and set the status filter to **Display Installed**.
2. Select the package or packages you wish to remove.
3. Right-click the selected packages and choose **Remove Selected Packages** from the shortcut menu.
4. Click **Next**; you will see the actions the **Package Manager** is about to carry out.
5. Click **Next** and the **Package Manager** will remove the package(s).
6. When the operation is complete, click **Finish** to close the **Package Manager**.

Reinstalling a package

The package-reinstall operation carries out a package-remove operation followed by a package-install operation.

To reinstall a package:

1. Choose **Tools** > **Package Manager** and set the status filter to **Display Installed**.
2. Select the package or packages you wish to reinstall.
3. Right-click the packages to reinstall and choose **Reinstall Selected Packages** from the shortcut menu.
4. Click **Next**; you will see the actions the **Package Manager** is about to carry out.
5. Click **Next** and the **Package Manager** will reinstall the packages.
6. When the operation is complete, click **Finish** to close the **Package Manager**.

Exploring your application

In this section, we discuss the SEGGER Embedded Studio tools that help you examine how your application is built.

Project explorer

The **Project Explorer** is the user interface of the SEGGER Embedded Studio project system. It organizes your projects and files and provides access to the commands that operate on them. A toolbar at the top of the window offers quick access to commonly used commands for the selected project node or the active project. Right-click to reveal a shortcut menu with a larger set of commands that will work on the selected project node, ignoring the active project.

The selected project node determines what operations you can perform. For example, the **Compile** operation will compile a single file if a file project node is selected; if a folder project node is selected, each of the files in the folder are compiled.

You can select project nodes by clicking them in the **Project Explorer**. Additionally, as you switch between files in the editor, the selection in the **Project Explorer** changes to highlight the file you're editing.

To activate the Project Explorer:

- Choose **View** > **Project Explorer** or press **Ctrl**+**Alt**+**P**.

Left-click operations

The following operations are available in the **Project Explorer** with a left-click of the mouse:

Action	Description
Single click	Select the node. If the node is already selected and is a solution, project, or folder node, a rename editor appears.
Double click	Double-clicking a solution node or folder node will reveal or hide the node's children. Double-clicking a project node selects it as the active project. Double-clicking a file opens the file with the default editor for that file's type.

Toolbar commands

The following buttons are on the toolbar:

Button	Description
	Add a new file to the active project using the **New File** dialog.
	Add existing files to the active project.

	Remove files, folders, projects, and links from the project.
	Create a new folder in the active project.
	Menu of build operations.
	Disassemble the active project.
	Menu of **Project Explorer** options.
	Display the properties dialog for the selected item.

Shortcut menu commands

The shortcut menu, displayed by right-clicking, contains the commands listed below.

For solutions:

Item	Description
Build and Batch Build	Build all projects under the solution in the current or batch build configuration.
Rebuild and Batch Rebuild	Rebuild all projects under the solution in the current or batch build configuration.
Clean and Batch Clean	Remove all output and intermediate build files for the projects under the solution in the current or batch build configuration.
Export Build and Batch Export Build	Create an editor with the build commands for the projects under the solution in the current or batch build configuration.
Add New Project	Add a new project to the solution.
Add Existing Project	Create a link from an existing solution to this solution.
Paste	Paste a copied project into the solution.
Remove	Remove the link to another solution from the solution.
Rename	Rename the solution node.
Source Control Operations	Source-control operations on the project file and recursive operations on all files in the solution.
Edit Solution As Text	Create an editor containing the project file.
Save Solution As	Change the filename of the project file—note that the saved project file is not reloaded.
Properties	Show the **Properties** dialog with the solution node selected.

For projects:

Item	Description
Build and Batch Build	Build the project in the current or batch build configuration.
Rebuild and Batch Rebuild	Reuild the project in the current or batch build configuration.
Clean and Batch Clean	Remove all output and intermediate build files for the project in the current or batch build configuration.
Export Build and Batch Export Build	Create an editor with the build commands for the project in the current or batch build configuration.
Link	Perform the project node build operation: link for an Executable project type, archive for a Library project type, and the combine command for a Combining project type.
Set As Active Project	Set the project to be the active project.
Debugging Commands	For Executable and Externally Built Executable project types, the following debugging operations are available on the project node: **Start Debugging**, **Step Into Debugging**, **Reset And Debug**, **Start Without Debugging**, **Attach Debugger**, and **Verify**.
Memory-Map Commands	For Executable project types that don't have memory-map files in the project and have the memory-map file project option set, there are commands to view the memory-map file and to import it into the project.
Section-Placement Commands	For Executable project types that don't have section-placement files in the project but have the section-placement file project option set, there are commands to view the section-placement file and to import it into the project.
Target Processor	For Executable and Externally Built Executable project types that have a Target Processor option group, the selected target can be changed.
Add New File	Add a new file to the project.
Add Existing File	Add an existing file to the project.
New Folder	Create a new folder in the project.
Cut	Cut the project from the solution.
Copy	Copy the project from the solution.
Paste	Paste a copied folder or file into the project.
Remove	Remove the project from the solution.
Rename	Rename the project.

Source Control Operations	Source-control, recursive operations on all files in the project.
Find in Project Files	Run **Find in Files** in the project directory.
Properties	Show the **Project Manager** dialog and select the project node.

For folders:

Item	Description
Add New File	Add a new file to the folder.
Add Existing File	Add an existing file to the folder.
New Folder	Create a new folder in the folder.
Cut	Cut the folder from the project or folder.
Copy	Copy the folder from the project or folder.
Paste	Paste a copied folder or file into the folder.
Remove	Remove the folder from the project or folder.
Rename	Rename the folder.
Source Control Operations	Source-control recursive operations on all files in the folder.
Compile	Compile each file in the folder.
Properties	Show the properties dialog with the folder node selected.

For files:

Item	Description
Open	Edit the file with the default editor for the file's type.
Open With	Edit the file with a selected editor. You can choose from the **Binary Editor**, **Text Editor**, and **Web Browser**.
Select in File Explorer	Create a operating system file system window with the file selected.
Compile	Compile the file.
Export Build	Create an editor window containing the commands to compile the file in the active build configuration.
Exclude From Build	Set the **Exclude From Build** option to **Yes** for this project node in the active build configuration.
Disassemble	Disassemble the output file of the compile into an editor window.
Preprocess	Run the C preprocessor on the file and show the output in an editor window.

Cut	Cut the file from the project or folder.
Copy	Copy the file from the project or folder.
Remove	Remove the file from the project or folder.
Import	Import the file into the project.
Source Control Operations	Source-control operations on the file.
Properties	Show the properties dialog with the file node selected.

Source navigator window

One of the best ways to find your way around your source code is using the **Source Navigator**. It parses the active project's source code and organizes classes, functions, and variables in various ways.

To activate the Source Navigator:

- Choose **View** > **Source Navigator** or press **Ctrl**+**Alt**+**N**.

The main part of the **Source Navigator** window provides an overview of your application's functions, classes, and variables.

SEGGER Embedded Studio displays these icons to the left of each object:

Icon	Description
	A C or C++ structure or a C++ namespace.
	A C++ class.
	A C++ member function declared `private` or a function declared with `static` linkage.
	A C++ member function declared `protected`.
	A C++ member function declared `public` or a function declared with `extern` linkage.
	A C++ member variable declared `private` or a variable declared with `static` linkage.
	A C++ member variable declared `protected`.
	A C++ member variable declared `public` or a variable declared with `extern` linkage.

Re-parsing after editing

The **Source Navigator** does not update automatically, only when you ask it to. To parse source files manually, click the **Refresh** button on the **Source Navigator** toolbar.

SEGGER Embedded Studio re-parses all files in the active project, and any dependent project, and updates the **Source Navigator** with the changes. Parsing progress is shown as a progress bar in the in the **Source Navigator** window. Errors and warnings detected during parsing are sent to the Source Navigator Log in the **Output** window—you can show the log quickly by clicking the **Show Source Navigator Log** tool button on the **Source Navigator** toolbar.

Sorting and grouping

You can group objects by their type; that is, whether they are classes, functions, namespaces, structures, or variables. Each object is placed into a folder according to its type.

To group objects by type:

1. On the **Source Navigator** toolbar, click the arrow to the right of the **Cycle Grouping** button.
2. Choose **Group By Type**

References window

The **References** window shows the results of the last **Find References** operation. The **Find References** facility is closely related to the **Source Navigator** in that it indexes your project and searches for references within the active source code regions.

To activate the References window:

If you have hidden the **References** window and want to see it again:

- Choose **View** > **References** or press **Ctrl+Alt+R**.

To find all references in a project:

1. Open a source file that is part of the active project, or one of its dependent projects.
2. In the editor, move the insertion point within the name of the function, variable, method, or macro to find.
3. Choose **Navigate** > **Find References** or press **Alt+R**.
4. SEGGER Embedded Studio shows the **References** window, without moving focus, and searches your project in the background.

You can also find references directly from the text editor's context menu: right-click the item to find and choose **Find References**. As a convenience, SEGGER Embedded Studio is configured to also run **Find References** when you Alt+Right-click in the text editor—see **Mouse-click accelerators**.

To search within the results:

- Type the text to search for in the Reference window's search box. As you type, the search results are narrowed.
- Click the close button to clear the search text and show all references.

To replace within the results:

- Type the replacement text in the Reference window's replace box.
- Use the buttons to navigate and replace the text.
- The documents that have had replaced text will appear unsaved in the text editor.

Symbol browser window

The **Symbol Browser** shows useful information about your linked application and complements the information displayed in the **Project Explorer** window. You can select different ways to filter and group the information in the **Symbol Browser** to provide an at-a-glance overview of your application. You can use the **Symbol Browser** to *drill down* to see the size and location of each part of your program. The way symbols are sorted and grouped is saved between runs; so, when you rebuild an application, SEGGER Embedded Studio automatically updates the **Symbol Browser** so you can see the effect of your changes on the memory layout of your program.

User interface

Button	Description
	Group symbols by source filename.
	Group symbols by symbol type (equates, functions, labels, sections, and variables).
	Group symbols by the section where they are defined.
	Move the insertion point to the statement that defined the symbol.
	Select columns to display.

The main part of the **Symbol Browser** displays each symbol (both external and static) that is linked into an application. SEGGER Embedded Studio displays the following icons to the left of each symbol:

Icon	Description
	*Private Equate* A private symbol not defined relative to a section.
	*Public Equate* A public symbol that is not defined relative to a section.
	*Private Function* A private function symbol.
	*Public Function* A public function symbol.
	*Private Label* A private data symbol, defined relative to a section.
	*Public Label* A public data symbol, defined relative to a section.
	*Section* A program section.

Choosing what to show

To activate the Symbol Browser window:

- Choose **View** > **Symbol Browser** or press **Ctrl**+**Alt**+**Y**.

You can choose to display the following fields for each symbol:

- *Value*: The value of the symbol. For labels, code, and data symbols, this will be the address of the symbol. For absolute or symbolic equates, this will be the value of the symbol.
- *Range*: The range of addresses the code or data item covers. For code symbols that correspond to high-level functions, the range is the range of addresses used for that function's code. For data addresses that correspond to high-level **static** or **extern** variables, the range is the range of addresses used to store that data item. These ranges are only available if the corresponding source file was compiled with debugging information turned on: if no debugging information is available, the range will simply be the first address of the function or data item.
- *Size*: The size, in bytes, of the code or data item. The **Size** column is derived from the **Range** of the symbol: if the symbol corresponds to a high-level code or data item and has a range, **Size** is calculated as the difference between the start and end addresses of the range. If a symbol has no range, the size column is blank.
- *Section*: The section in which the symbol is defined. If the symbol is not defined within a section, the **Section** column is blank.
- *Type*: The high-level type for the data or code item. If the source file that defines the symbol is compiled with debugging information turned off, type information is not available and the **Type** column is blank.
- *Frame Size*: The amount of stack space used by a call to the function symbol. If the source file that defines the symbol is compiled with debugging information turned off, frame size information is not available and the **Type** column is blank.

Initially the **Range** and **Size** columns are shown in the **Symbol Browser**. To select which columns to display, use the **Field Chooser** button on the **Symbol Browser** toolbar.

To select the fields to display:

1. Click the **Field Chooser** button on the **Symbol Browser** toolbar.
2. Select the fields you wish to display and deselect the fields you wish to hide.

Organizing and sorting symbols

When you group symbols by section, each symbol is grouped underneath the section in which it is defined. Symbols that are absolute or are not defined within a section are grouped beneath '(No Section)'.

To group symbols by section:

1. On the **Symbol Browser** toolbar, click the arrow next to the **Cycle Grouping** button.

2. From the pop-up menu, choose **Group By Section**.

The **Cycle Grouping** icon will change to indicate that the **Symbol Browser** is grouping symbols by section.

When you group symbols by type, each symbol is classified as one of the following:

- An *Equate* has an absolute value and is not defined as relative to, or inside, a section.
- A *Function* is defined by a high-level code sequence.
- A *Variable* is defined by a high-level data declaration.
- A *Label* is defined by an assembly language module. *Label* is also used when high-level modules are compiled with debugging information turned off.

When you group symbols by source file, each symbol is grouped underneath the source file in which it is defined. Symbols that are absolute, are not defined within a source file, or are compiled without debugging information, are grouped beneath '(Unknown)'.

To group symbols by type:

1. On the **Symbol Browser** toolbar, click the arrow next to the **Cycle Grouping** button.
2. Choose **Group By Type** from the pop-up menu.

The **Cycle Grouping** icon will change to indicate that the **Symbol Browser** is grouping symbols by type.

To group symbols by source file:

1. On the **Symbol Browser** toolbar, click the arrow next to the **Cycle Grouping** button.
2. Choose **Group By Source File**.

The **Cycle Grouping** icon will change to indicate that the **Symbol Browser** is grouping symbols by source file.

When you sort symbols alphabetically, all symbols are displayed in a single list in alphabetical order.

To list symbols alphabetically:

1. On the **Symbol Browser** toolbar, click the arrow next to the **Cycle Grouping** button.
2. Choose **Sort Alphabetically**.

The **Cycle Grouping** icon will change to indicate that the **Symbol Browser** is grouping symbols alphabetically.

Filtering and finding symbols

When you're dealing with big projects with hundreds, or even thousands, of symbols, a way to filter those symbols in order to isolate just the ones you need is very useful. The **Symbol Browser**'s toolbar provides an editable *combo*box} you can use to specify the symbols you'd like displayed. You can type '*' to match a sequence of zero or more characters and '?' to match exactly one character.

The symbols are filtered and redisplayed as you type into the combo box. Typing the first few characters of a symbol name is usually enough to narrow the display to the symbol you need. *Note:* the C compiler prefixes all high-level language symbols with an underscore character, so the variable `extern int u` or the function `void fn(void)` have low-level symbol names _u and _fn. The **Symbol Browser** uses the low-level symbol name when displaying and filtering, so you must type the leading underscore to match high-level symbols.

To display symbols that start with a common prefix:

- Type the desired prefix text into the combo box, optionally followed by a "*".

For instance, to display all symbols that start with "**i2c**_", type "**i2c**_" and all matching symbols are displayed—you don't need to add a trailing "*" in this case, because it is implied.

To display symbols that end with a common suffix:

- Type '*' into the combo box, followed by the required suffix.

For instance, to display all symbols that end in '_data', type '*_data' and all matching symbols are displayed—in this case, the leading '*' is required.

When you have found the symbol you're interested in and your source files have been compiled with debugging information turned on, you can jump to a symbol's definition using the **Go To Definition** button.

To jump to the definition of a symbol:

1. Select the symbol from the list of symbols.
2. On the **Symbol Browser** toolbar, click **Go To Definition**.

—or—

1. Right-click the symbol in the list of symbols.
2. Choose **Go To Definition** from the shortcut menu.

Watching symbols

If a symbol's range and type is known, you can add it to the most recently opened **Watch** window or **Memory** window.

To add a symbol to the Watch window:

1. In the **Symbol Browser**, right-click the symbol you wish to add to the **Watch** window.
2. On the shortcut menu, choose **Add To Watch**.

To add a symbol to the Memory window:

1. In the **Symbol Browser**, right-click the symbol you wish to add to the **Memory** window.

2. Choose **Locate Memory** from the shortcut menu.

Using size information

Here are a few common ways to use the **Symbol Browser**:

What function uses the most code space? What requires the most data space?

1. Choose **View** > **Symbol Browser** or press **Ctrl+Alt+Y**.
2. In the **Grouping** button menu on the **Symbol Browser** toolbar, select **Group By Type**.
3. Ensure the **Size** field is checked in the **Field Chooser** button's menu.
4. Ensure that the filter on the **Symbol Browser** toolbar is empty.
5. Click on the **Size** field in the header to sort by data size.
6. The sizes of variables and of functions are shown in separate lists.

What's the overall size of my application?

1. Choose **View** > **Symbol Browser** or press **Ctrl+Alt+Y**.
2. In the **Grouping** button menu on the **Symbol Browser** toolbar, select **Group By Section**.
3. Ensure the **Range** and **Size** fields are checked in the **Field Chooser** button's menu.
4. Read the section sizes and ranges of each section in the application.

Stack usage window

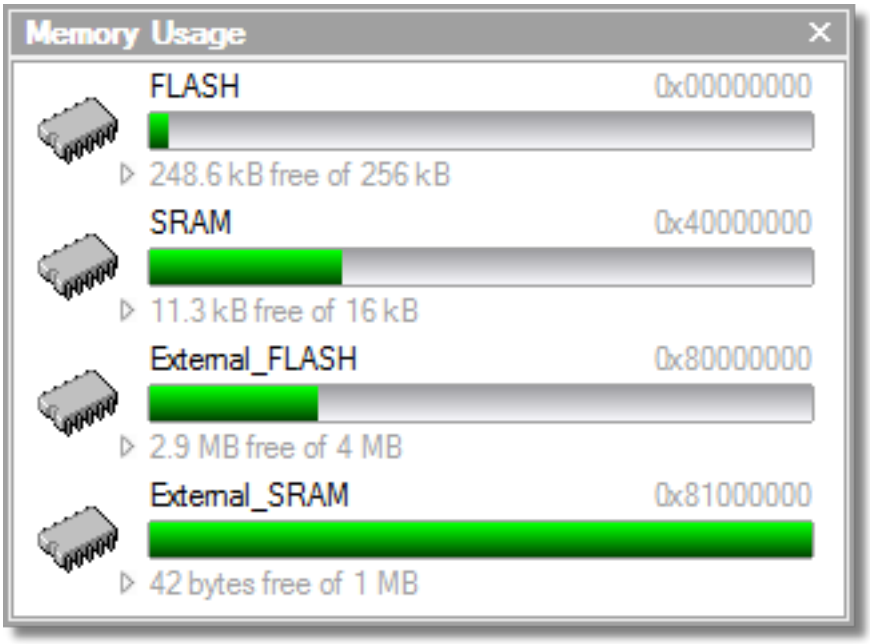
The **Stack Usage Window** finds the call paths of your linked application and displays them as a call tree together with their minimal stack requirements. A call path of your application is any function that has been linked in but has no direct call made to it but will make calls to other functions. The main function is the most obvious example of a call path, an interrupt handler or a function that is called only as a function pointer are other examples. To use the stack usage window your linked application must be compiled with debugging information enabled.

User interface

Button	Description
	Move the insertion point to the statement that defined the symbol.
	Collapse the selected open call tree.
	Open the selected open call tree.
	Show only the deepest call path through the selected call tree.

Memory usage window

The **Memory Usage** window displays a graphical summary of how memory has been used in each memory segment of a linked application.



Each bar represents an entire memory segment. Green represents the area of the segment that contains code or data.

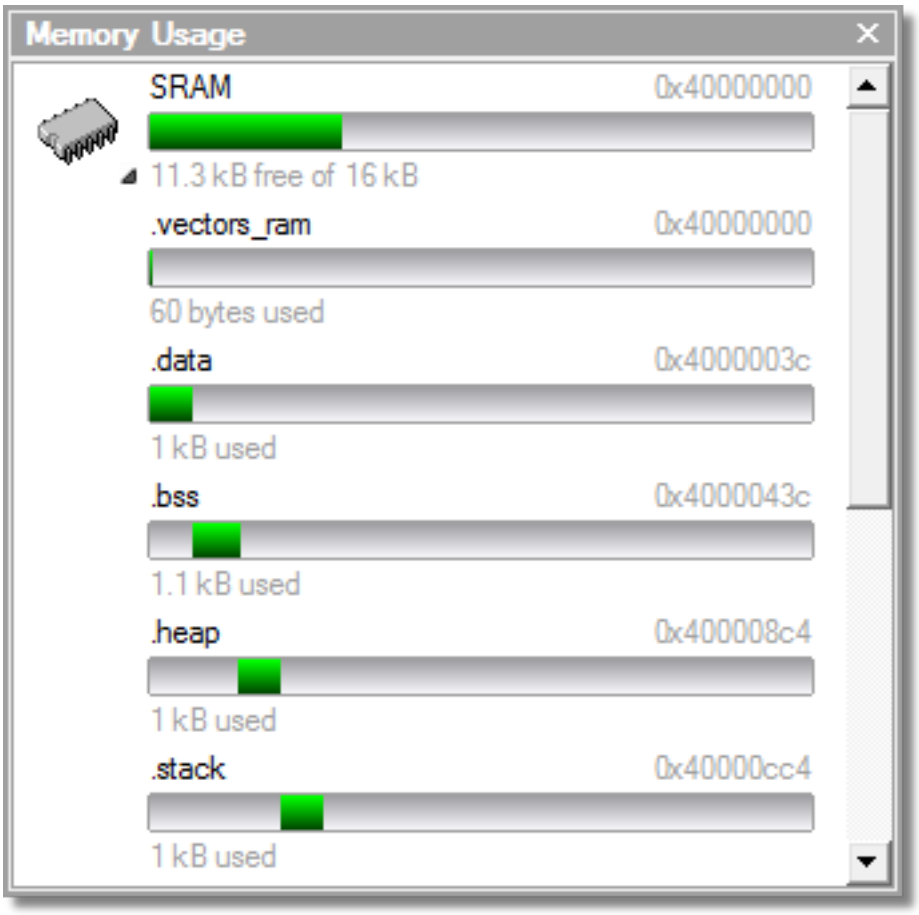
To activate the Memory Usage window:

- Choose **View** > **Memory Usage** or press **Ctrl**+**Alt**+**Z**.

The memory-usage graph will only be visible if your active project's target is an executable file and the file exists. If the executable file has not been linked by SEGGER Embedded Studio, memory-usage information may not be available.

Displaying section information

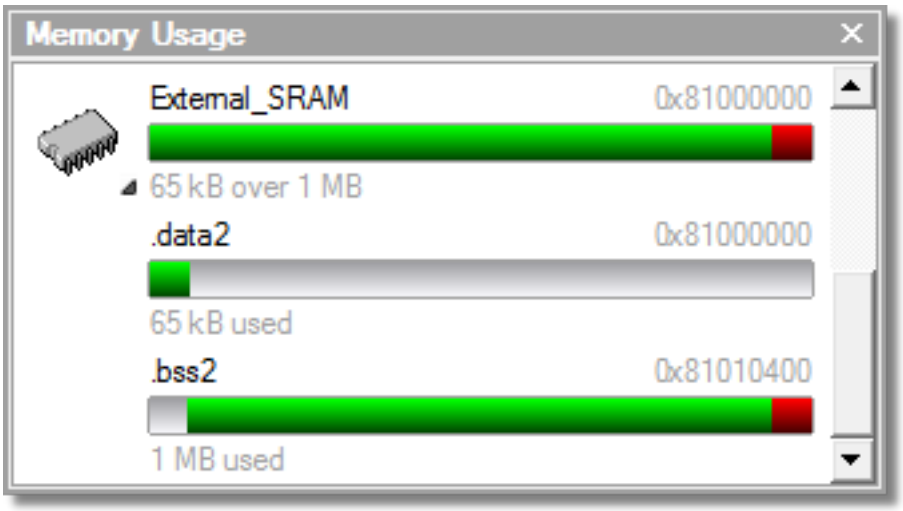
The **Memory Usage** window can also be used to visualize how program sections have been placed in memory. To display the program sections, simply click the memory segment to expand it; or, alternatively, right-click and choose **Show Memory Sections** from the shortcut menu.



Each bar represents an entire memory segment. Green represents the area of the segment that contains the program section.

Displaying segment overflow

The **Memory Usage** window also displays segment overflows when the total size of the program sections placed in a segment is larger than the segment size. When this happens, the segment and section bars represents the total memory used, green areas represent the code or data within the segment, and red areas represent code or data placed outside the segment.



Getting more-detailed information

If you require more-detailed information than that provided by the **Memory Usage** window, such as the location of specific objects within memory, use the **Symbol browser window**.

Bookmarks window

The **Bookmarks** window contains a list of bookmarks that are set in the project. The bookmarks are stored in the session file associated with the project and persist across runs of SEGGER Embedded Studio—if you remove the session file, the bookmarks associated with the project are lost.

User interface

Button	Description
	Toggle a bookmark at the insertion point in the active editor. Equivalent to choosing **Edit** > **Bookmarks** > **Toggle Bookmark** or pressing **Ctrl+F2**.
	Go to the previous bookmark in the bookmark list. Equivalent to choosing **Edit** > **Bookmarks** > **Previous Bookmark** or pressing **Alt+Shift+F2**.
	Go to the next next bookmark in the bookmark list. Equivalent to choosing **Edit** > **Bookmarks** > **Next Bookmark** or pressing **Alt+F2**.
	Clear all bookmarks—you confirm the action using a dialog. Equivalent to choosing **Edit** > **Bookmarks** > **Clear All Bookmarks** or pressing **Ctrl+K, Alt+F2**.
	Selects the fill color for newly created bookmarks.

Double-clicking a bookmark in the bookmark list moves focus to the the bookmark.

You can set bookmarks with the mouse or using keystrokes—see **Using bookmarks**.

Code Outline Window

The **Code Outline** window shows the structure of the text of the focused code editor. For C and C++ documents the top level symbols and types are displayed, for XML documents the nodes are displayed.

Analyzing Source Code

The **Analyze** action is available on the context menu of the project explorer at project, folder and file level. The analyze action will run the **https://clang.llvm.org/extra/clang-tidy** linter tool on the C/C++ files selected by the project explorer node and display warnings in the output window. The default checks will be the same as the clang analyzer. You can enable additional checks by setting the **Clang Tidy Checks** project option. For example you can enable the bugprone code constructs check and disable a specific clang analyzer diagnostic check as follows

```
bugprone-*
-clang-diagnostic-parentheses-equality
```

You can also set the project option **Analyze After Compile** which will run the analyzer each time the compiler is run.

Editing your code

SEGGER Embedded Studio has a built-in editor that allows you to edit text, but some features make it particularly well suited to editing code.

You can open multiple code editors to browse or edit project source code, and you can copy and paste among them. The **Windows** menu contains a list of all open code editors.

The code editor supports the language of the source file it is editing, showing code with syntax highlighting and offering smart indenting.

You can open a code editor in several ways, some of which are:

- By double-clicking a file in the **Project Explorer** or by right-clicking a file and selecting **Open** from the shortcut menu.
- Using the **File** > **New File** or **File** > **Open** commands.

Elements of the code editor

The code editor is composed of several elements, which are described here.

- *Code pane*: The area where you edit code. You can set options that affect the code pane's text indents, tabs, drag-and-drop behavior, and so forth.
- *Margin gutter*: A gray area on the left side of the code editor where margin indicators such as breakpoints, bookmarks, and shortcuts are displayed. Clicking this area sets a breakpoint on the corresponding line of code.
- *Horizontal and vertical scroll bars*: You can scroll the code pane horizontally and vertically to view code that extends beyond the edges of the pane.

Basic editing

This section is a whirlwind tour of the basic editing features SEGGER Embedded Studio's code editor provides.

Whether you are editing code, HTML, or plain text, the code editor is just like many other text editors or word processors. For code that is part of a project, the project's programming language support provides syntax highlighting (colorization), indentation, and so on.

This section *is not* a reference for everything the code editor provides; for that, look in the following sections.

Moving the insertion point

The most common way to navigate through text is to use use the mouse or the keyboard's cursor keys.

Using the mouse

You can move the insertion point within a document by clicking the mouse inside the editor window.

Using the keyboard

The keystrokes most commonly used to navigate through a document are:

Keystroke	Description
Up	Move the insertion point up one line
Down	Move the insertion point down one line
Left	Move the insertion point left one character
Right	Move the insertion point right one character
Home	Move the insertion point to the first non-whitespace character on the line — pressing **Home** a second time moves the insertion point to the leftmost column
End	Move the insertion point to the end of the line
PageUp	Move the insertion point up one page
PageDown	Move the insertion point down one page
Ctrl+Home	Move the insertion point to the start of the document
Ctrl+End	Move the insertion point to the end of the document
Ctrl+Left	Move the insertion point left one word
Ctrl+Right	Move the insertion point right one word

SEGGER Embedded Studio offers additional movement keystrokes, though most users are more comfortable using repeated simple keystrokes to accomplish the same thing:

Keystroke	Description
Alt+Up	Move the insertion point up five lines
Alt+Down	Move the insertion point down five lines
Alt+Home	Move the insertion point to the top of the window
Alt+End	Move the insertion point to the bottom of the window
Ctrl+Up	Scroll the document up one line in the window without moving the insertion point

Ctrl+Down	Scroll the document down one line in the window without moving the insertion point

If you are editing source code, the are source-related keystrokes too:

Keystroke	Description
Ctrl+PgUp	Move the insertion point backwards to the previous function or method.
Ctrl+PgDn	Move the insertion point forwards to the next function or method.

Adding text

The editor has two text-input modes:

- *Insertion mode*: As you type on the keyboard, text is entered at the insertion point and any text to the right of the insertion point is shifted along. A visual indication of insertion mode is that the cursor is a flashing line.
- *Overstrike mode*: As you type on the keyboard, text at the insertion point is replaced with your typing. A visual indication of insertion mode is that the cursor is a flashing block.

Insert and overstrike modes are common to *all* editors: if one editor is in insert mode, *all* editors are in insert mode. To configure the cursor appearance, choose **Tools** > **Options**.

To toggle between insertion and overstrike mode:

- Click **Insert**.

When overstrike mode is enabled, the mode indicator changes from **INS** to **OVR** and the cursor will change to the overstrike cursor.

To add or insert text:

1. Move the insertion point to the place text is to be inserted.
2. Enter the text using the keyboard.

To overwrite characters in an existing line, press the **Insert** key to place the editor into overstrike mode.

To add or insert text on multiple lines:

1. Hold down the **Alt** key and use block selection to mark the place text is to be inserted.
2. Enter the text using the keyboard.

Deleting text

The text editor supports the following common editing keystrokes:

Keystroke	Description
Backspace	Delete the character before the insertion point
Delete	Delete the character after the insertion point
Ctrl+Backspace	Delete one word before the insertion point
Ctrl+Delete	Delete one word after the insertion point

To delete characters or words:

1. Place the insertion point before the word or letter you want to delete.
2. Press **Delete** as many times as needed.

—or—

1. Place the insertion point after the letter or word you want to delete.
2. Press **Backspace** as many times as needed.

To delete text that spans more than a few characters:

1. Select the text you want to delete.
2. Press **Delete** or **Backspace** to delete it.

To delete a text block:

1. Hold down the **Alt** key and use block selection to mark the text you want to delete.
2. Press **Delete** or **Backspace** to delete it.

To delete characters on multiple lines:

1. Hold down the **Alt** key and use block selection to mark the lines.
2. Press **Delete** or **Backspace** as many times as needed to delete the characters.

Using the clipboard

You can select text by using the keyboard or the mouse.

To select text with the keyboard:

- Hold down the **Shift** key while using the cursor keys.

To select text with the mouse:

1. Click the start of the selection.
2. Drag the mouse to mark the selection.
3. Release the mouse to end selecting.

To select a block of text with the keyboard:

- Hold down the **Shift+Alt** keys while using the cursor keys.

To select a block of text with the mouse:

1. Hold down the **Alt** key.
2. Click the start of the selection.
3. Drag the mouse to mark the selection.
4. Release the mouse to end selecting.

To copy selected text to the clipboard:

- Choose **Edit** > **Copy** or press **Ctrl+C**.

The standard Windows key sequence **Ctrl+Ins** also copies text to the clipboard.

To cut selected text to the clipboard:

- Choose **Edit** > **Cut** or press **Ctrl+X**.

The standard Windows key sequence **Shift+Del** also cuts text to the clipboard.

To insert the clipboard content at the insertion point:

- Choose **Edit** > **Paste** or press **Ctrl+V**.

The standard Windows key sequence **Shift+Ins** also inserts the clipboard content at the insertion point.

Undo and redo

The editor has an *undo* facility to undo previous editing actions. The *redo* feature can be used to re-apply previously undone actions.

To undo one editing action:

- Choose **Edit** > **Undo** or press **Ctrl**+**Z**.

The standard Windows key sequence **Alt**+**Backspace** also undoes an edit.

To undo multiple editing actions:

1. On the **Standard** toolbar, click the arrow next to the **Undo** button.
2. Select the editing operations to undo.

To undo all edits:

- Choose **Edit** > **Others** > **Undo All** or press **Ctrl**+**K, Ctrl**+**Z**.

To redo one editing action:

- Choose **Edit** > **Redo** or press **Ctrl**+**Y**.

The standard Windows key sequence **Alt**+**Shift**+**Backspace** also redoes an edit.

To redo multiple editing actions:

1. On the **Standard** toolbar, click the arrow next to the **Redo** tool button.
2. From the pop-up menu, select the editing operations to redo.

To redo all edits:

- Choose **Edit** > **Others** > **Redo All** or press **Ctrl**+**K, Ctrl**+**Y**.

Drag and drop

You can select text, then drag it to another location. You can drop the text at a different location in the same window or in another one.

To drag and drop text:

1. Select the text you want to move.
2. Press and hold the mouse button to drag the selected text to where you want to place it.
3. Release the mouse button to drop the text.

Dragging text *moves* it to the new location. To *copy* it to a new location, hold down the **Ctrl** key while dragging the text: the mouse pointer changes to indicate a copy operation. Press the **Esc** key while dragging text to cancel the drag-and-drop edit.

By default, drag-and drop-editing is *disabled* and you must enable it if you want to use it.

To enable or disable drag-and-drop editing:

1. Choose **Tools** > **Options** or press **Alt+,**.
2. Click **Text Editor**.
3. Set **Allow Drag and Drop Editing** to **Yes** to enable or to **No** to disable drag-and-drop editing.

Searching

To find text in the current file:

1. Press **Ctrl+F**.
2. Enter the string to search for.

As you type, the editor searches the file for a match. The pop-up shows how many matches are in the current file. To move through the matches while the **Find** box is still active, press **Tab** or **F3** to move to the next match and **Shift+Tab** or **Shift+F3** to move to the previous match.

If you press **Ctrl+F** a second time, SEGGER Embedded Studio pops up the standard **Find** dialog to search the file. If you wish to bring up the **Find** dialog without pressing `Ctrl+F` twice, choose **Search** > **Find**.

Advanced editing

You can do anything using its basic code-editing features, but the SEGGER Embedded Studio text editor has a host of labor-saving features that make editing programs a snap.

This section describes the code-editor features intended to make editing source code easier.

Indenting source code

The editor uses the **Tab** key to increase or decrease the indentation level of the selected text.

To increase indentation:

- Select the text to indent.
- Choose **Selection** > **Increase Line Indent** or press **Tab**.

To decrease indentation:

- Select the text to indent.
- Choose **Selection** > **Decrease Line Indent** or press **Shift+Tab**.

The indentation size can be changed in the **Language Properties** pane of the editor's **Properties** window, as can all the indent-related features listed below.

To change indentation size:

- Choose **Tools** > **Options** or press **Alt+,**.
- Select the **Languages** page.
- Set the **Indent Size** environment option for the required language.

You can choose to use spaces or tab tab characters to fill whitespace when indenting.

To set tab or space fill when indenting:

- Choose **Tools** > **Options** or press **Alt+,**.
- Select the **Languages** page.
- Set the **Use Tabs** environment option for the required language. *Note:* changing this setting does not add or remove existing tabs from files, the change will only affect new indents.

The editor can assist with source code indentation while inserting text. There are three levels of indentation assistance:

- *None*: The indentation of the source code is left to the user.
- *Indent*: This is the default. The editor maintains the current indentation level. When you press **Return** or **Enter**, the editor moves the insertion point down one line and indented to the same level as the now-previous line.
- *Smart*: The editor analyzes the source code to compute the appropriate indentation level for each line. You can change how many lines before the insertion point will be analyzed for context. The smart-indent mode can be configured to indent either open and closing braces or the lines following the braces.

Changing indentation options:

To change the indentation mode:

- Set the **Indent Mode** environment option for the required language.

To change whether opening braces are indented in smart-indent mode:

- Set the **Indent Opening Brace** environment option for the required language.

To change whether closing braces are indented in smart-indent mode:

- Set the **Indent Closing Brace** environment option for the required language.

To change the number of previous lines used for context in smart-indent mode:

- Set the **Indent Context Lines** environment option for the required language.

Commenting out sections of code

To comment selected text:

- Choose **Selection** > **Comment** or press **Ctrl**+/.

To uncomment selected text:

- Choose **Selection** > **Uncomment** or press **Ctrl**+**Shift**+/.

You can also toggle the commenting of a selection by typing /. This has no menu equivalent.

Adjusting letter case

The editor can change the case of the current word or the selection. The editor will change the case of the selection, if there is a selection, otherwise it will change the case of word at the insertion point.

To change text to uppercase:

- Choose **Selection** > **Make Uppercase** or press **Ctrl**+**K, U**.

This changes, for instance, 'Hello' to 'HELLO'.

To change text to lowercase:

- Choose **Selection** > **Make Lowercase** or press **Ctrl**+**U**.

This changes, for instance, 'Hello' to 'hello.'

To switch between uppercase and lowercase:

- Choose **Selection** > **Switch Case**.

This changes, for instance, 'Hello' to 'hELLO.'

With large software teams or imported source code, sometimes identifiers don't conform to your local coding style. To assist in conversion between two common coding styles for identifiers, SEGGER Embedded Studio's editor offers the following two shortcuts:

To change from split case to camel case:

- Choose **Selection** > **Camel Case** or press **Ctrl**+**K, Ctrl**+**Shift**+**U**.

This changes, for instance, 'this_is_wrong' to 'thisIsWrong.'

To change from camel case to split case:

- Choose **Selection** > **Split Case** or press **Ctrl**+**K, Ctrl**+**U**.

This changes, for instance, 'thisIsWrong' to 'this_is_wrong.'

Using bookmarks

To edit a document elsewhere and then return to your current location, add a bookmark. The **Bookmarks** window maintains a list of the bookmarks set in source files — see **Bookmarks window**.

To place a bookmark:

1. Move the insertion point to the line you wish to bookmark.
2. Choose **Edit** > **Bookmarks** > **Toggle Bookmark** or press **Ctrl+F2**.

A bookmark symbol appears next to the line in the indicator margin to show the bookmark is set.

To place a bookmark using the mouse:

1. Right-click the margin gutter where the bookmark should be set.
2. Choose **Toggle Bookmark**.

The default color to use for new bookmarks is configured in the **Bookmarks** window. You can choose a specific color for the bookmark as follows:

1. Press and hold the **Alt** key.
2. Click the margin gutter where the bookmark should be set.
3. From the palette, click the bookmark color to use for the bookmark.

To navigate forward through bookmarks:

1. Choose **Edit** > **Bookmarks** > **Next Bookmark In Document** or press **F2**.
2. The editor moves the insertion point to the next bookmark in the document.

If there is no following bookmark, the insertion point moves to the first bookmark in the document.

To navigate backward through bookmarks:

1. Choose **Edit** > **Bookmarks** > **Previous Bookmark In Document** or press **Shift+F2**.
2. The editor moves the insertion point to the previous bookmark in the document.

If there is no previous bookmark, the insertion point moves to the last bookmark in the document.

To remove a bookmark:

1. Move the insertion point to the line containing the bookmark.
2. Choose **Edit** > **Bookmarks** > **Toggle Bookmark** or press **Ctrl+F2**.

The bookmark symbol disappears, indicating the bookmark is no longer set.

To remove all bookmarks in a document:

- Choose **Edit** > **Bookmarks** > **Clear Bookmarks In Document** or press **Ctrl+K, F2**.

Quick reference for bookmark operations

Keystroke	Menu	Description
Ctrl+F2	**Edit > Bookmarks > Toggle Bookmark**	Toggle a bookmark at the insertion point.
Ctrl+K, 0		Clear the bookmark at the insertion point.
F2	**Edit > Bookmarks > Next Bookmark In Document**	Move the insertion point to next bookmark in the document.
Shift+F2	**Edit > Bookmarks > Previous Bookmark In Document**	Move the insertion point to previous bookmark in the document.
Ctrl+Q, F2	**Edit > Bookmarks > First Bookmark In Document**	Move the insertion point to the first bookmark in the document.
Ctrl+Q, Shift+F2	**Edit > Bookmarks > Last Bookmark In Document**	Move the insertion point to the last bookmark in the document.
Ctrl+K, F2	**Edit > Bookmarks > Clear Bookmarks In Document**	Clear all bookmarks in the document.
Alt+F2	**Edit > Bookmarks > Next Bookmark**	Move the insertion point to the next bookmark in the Bookmarks list.
Alt+Shift+F2	**Edit > Bookmarks > Previous Bookmark**	Move the insertion point to the previous bookmark in the Bookmarks list.
Ctrl+Q, Alt+F2	**Edit > Bookmarks > First Bookmark**	Move the insertion point to the first bookmark in the Bookmarks list.
Ctrl+Q, Alt+Shift+F2	**Edit > Bookmarks > Last Bookmark**	Move the insertion point to the last bookmark in the Bookmarks list.
Ctrl+K, Alt+F2	**Edit > Bookmarks > Clear All Bookmarks**	Clear all bookmarks in all documents.

Find and Replace window

The **Find and Replace** window allows you to search for and replace text in the current document or in a range of specified files.

To activate the Find and Replace window:

- Choose **Search** > **Replace in Files** or press **Ctrl+Alt+F**.

To find text in a single file:

- Select **Current Document** in the context combo box.
- Enter the string to be found in the text edit input.
- If the search will be case sensitive, set the **Match case** option.
- If the search will be for a whole word—i.e., there will be whitespace, such as spaces or the beginning or end of the line, on both sides of the string being searched for—set the **Whole word** option.
- If the search string is a regular expression, set the **Use regexp** option.
- Click the **Find** button to find all occurrences of the string in the current document.

To find and replace text in a single file:

- Click the **Replace** button on the toolbar.
- Enter the string to search for into the **Find what** input.
- Enter the replacement string into the **Replace with** input. If the search string is a regular expression, the *n* back-reference can be used in the replacement string to reference captured text.
- If the search will be case sensitive, set the **Match case** option.
- If the search will be for a whole word—i.e., there will be whitespace, such as spaces or the beginning or end of the line, on both sides of the string being searched for—set the **Match whole word** option.
- If the search string is a regular expression, set the **Use regular expression** option.
- Click the **Find Next** button to find next occurrence of the string, then click the **Replace** button to replace the found string with the replacement string; or click **Replace All** to replace all occurrences of the search string without prompting.

To find text in multiple files:

- Click the **Find In Files** button on the toolbar.
- Enter the string to search for into the **Find what** input.
- Select the appropriate option in the **Look in** input to select whether to carry out the search in all open documents, all documents in the current project, all documents in the current solution, or all files in a specified folder.
- If you have specified that you want to search in a folder, select the folder you want to search by entering its path in the **Folder** input and use the **Look in files matching** input to specify the type of files you want to search.

- If the search will be case sensitive, set the **Match case** option.
- If the search will be for a whole word—i.e., there will be whitespace, such as spaces or the beginning or end of the line, on both sides of the string being searched for—set the **Match whole word** option.
- If the search string is a regular expression, set the **Use regular expression** option.
- Click the **Find All** button to find all occurrences of the string in the specified files, or click the **Bookmark All** button to bookmark all the occurrences of the string in the specified files.

To replace text in multiple files:

- Click the **Replace In Files** button on the toolbar.
- Enter the string to search for into the **Find what** input.
- Enter the replacement string into the **Replace with** input. If the search string is a regular expression, the *n* back-reference can be used in the replacement string to reference captured text.
- Select the appropriate option in the **Look in** input to select whether you want to carry out the search and replace in the current or in all open documents.
- If you have specified that you want to search in a folder, select the folder you want to search by entering its path in the **Folder** input and use the **Look in files matching** input to specify the type of files you want to search.
- If the search will be case sensitive, set the **Match case** option.
- If the search will be for a whole word—i.e., there will be whitespace, such as spaces or the beginning or end of the line, on both sides of the string being searched for—set the **Match whole word** option.
- If the search string is a regular expression, set the **Use regular expression** option.
- Click the **Replace All** button to replace all occurrences of the string in the specified files.

Clipboard Ring window

The code editor captures all cut and copy operations, and stores the cut or copied item on the *clipboard ring*. The clipboard ring stores the last 20 cut or copied text items, but you can configure the maximum number by using the environment options dialog. The clipboard ring is an excellent place to store scraps of text when you're working with many documents and need to cut and paste between them.

To activate the clipboard ring:

- Choose **Edit** > **Clipboard Ring** > **Clipboard Ring** or press **Ctrl+Alt+C**.

To paste from the clipboard ring:

1. Cut or copy some text from your code. The last item you cut or copy into the clipboard ring is the current item for pasting.
2. Press **Ctrl+Shift+V** to paste the clipboard ring's current item into the current document.
3. Repeatedly press **Ctrl+Shift+V** to cycle through the entries in the clipboard ring until you get to the one you want to permanently paste into the document. Each time you press **Ctrl+Shift+V**, the editor replaces the last entry you pasted from the clipboard ring, so you end up with just the last one you selected. The item you stop on then becomes the current item.
4. Move to another location or cancel the selection. You can use **Ctrl+Shift+V** to paste the current item again or to cycle the clipboard ring to a new item.

Clicking an item in the clipboard ring makes it the current item.

To paste a specific item from the clipboard ring:

1. Move the insertion point to the position to paste the item in the document.
2. Click the arrow at the right of the item to paste.
3. Choose *Paste* from the pop-up menu.

—or—

1. Click the item to paste to make it the current item.
2. Move the insertion point to the position to paste the item in the document.
3. Press **Ctrl+Shift+V**.

To paste all items into a document:

To paste all items on the clipboard ring into the current document, move the insertion point to where you want to paste the items and do one of the following:

- Choose **Edit** > **Clipboard Ring** > **Paste All**.

—or—

- On the **Clipboard Ring** toolbar, click the **Paste All** button.

To remove an item from the clipboard ring:

1. Click the arrow at the right of the item to remove.
2. Choose **Delete** from the pop-up menu.

To remove all items from the clipboard ring:

- Choose **Edit** > **Clipboard Ring** > **Clear Clipboard Ring**.

—or—

- On the **Clipboard Ring** toolbar, click the **Clear Clipboard Ring** button.

To configure the clipboard ring:

1. Choose **Tools** > **Options** or press **Alt+,**.
2. Click the **Windows** category to show the **Clipboard Ring Options** group.
3. Select **Preserve Contents Between Runs** to save the content of the clipboard ring between runs, or deselect it to start with an empty clipboard ring.
4. Change **Maximum Items Held In Ring** to configure the maximum number of items stored on the clipboard ring.

Mouse-click accelerators

SEGGER Embedded Studio provides a number of mouse-click accelerators in the editor that speed access to commonly used functions. The mouse-click accelerators are user configurable using **Tools** > **Options**.

Default mouse-click assignments

Click	Default
Left	Not configurable — start selection.
Shift+Left	Not configurable — extend selection.
Ctrl+Left	Select word.
Alt+Left	Execute **Go To Definition**.
Middle	No action.
Shift+Middle	Display **Go To Include** menu.
Ctrl+Middle	No action.
Alt+Middle	Display **Go To Method** menu.
Right	Not configurable — show context menu.
Shift+Right	No action.
Ctrl+Right	No action.
Alt+Right	Execute **Find References**.

Each accelerator can be assigned one of the following actions:

- *Default*: The system default for that click.
- *Go To Definition*: Go to the definition of the item clicked, equivalent to choosing **Navigate** > **Go To Definition** or pressing **Alt+G**.
- *Find References*: Find references to the item clicked, equivalent to choosing **Navigate** > **Find References** or pressing **Alt+R**.
- *Find in Solution*: Textually find the item clicked in all the files in the solution, equivalent to choosing **Search** > **Find Extras** > **Find in Solution** or pressing **Alt+U**.
- *Find Help*: Use F1-help on the item clicked, equivalent to choosing **Help** > **Help** or pressing **F1**.
- *Go To Method*: Display the **Go To Method** menu, equivalent to choosing **Navigate** > **Find Method** or pressing **Ctrl+M**.
- *Go To Include*: Display the **Go To Include** menu, equivalent to choosing **Navigate** > **Find Include** or pressing **Ctrl+Shift+M**.
- *Paste*: Paste the clipboard at the position clicked, equivalent to choosing **Edit** > **Paste** or pressing **Ctrl+V**.

Configuring Mac OS X

On Mac OS X you must configure the mouse to pass middle clicks and right clicks to the application if you wish to use mouse-click accelerators in SEGGER Embedded Studio. Configure the mouse preferences in the **Mouse** control panel in Mac OS X **System Preferences** to the following:

- Right mouse button set to **Secondary Button**.
- Middle mouse button set to **Button 3**.

Regular expressions

The editor can search and replace text using *regular expressions*. A regular expression is a string that uses special characters to describe and reference patterns of text. The regular expression system used by the editor is modeled on Perl's regexp language. For more information on regular expressions, see *Mastering Regular Expressions*, Jeffrey E F Freidl, ISBN 0596002890.

Summary of special characters

The following table summarizes the special characters the SEGGER Embedded Studio editor supports

Pattern	Description
\d	Match a numeric character.
\D	Match a non-numeric character.
\s	Match a whitespace character.
\S	Match a non-whitespace character.
\w	Match a word character.
\W	Match a non-word character.
[*c*]	Match set of characters; e.g., [ch] matches characters c or h. A range can be specified using the '-' character; e.g., '[0-27-9]' matches if the character is 0, 1, 2, 7 8, or 9. A range can be negated using the '^' character; e.g., '[^a-z]' matches if the character is anything other than a lowercase alphabetic character.
c	Match the literal character *c*. For example, you would use '*' to match the character '*'.
\a	Match ASCII bell character (ASCII code 7).
\f	Match ASCII form feed character (ASCII code 12).
\t	Match ASCII horizontal tab character (ASCII code 9).
\v	Match ASCII vertical tab character.
\x*hhhh*	Match Unicode character specified by hexadecimal number *hhhh*.
.	Match any character.
*	Match zero or more occurrences of the preceding expression.
+	Match one or more occurrences of the preceding expression.
?	Match zero or one occurrences of the preceding expression.

{*n*}	Match *n* occurrences of the preceding expression.
{*n*,}	Match at least *n* occurrences of the preceding expression.
{,*m*}	Match at most *m* occurrences of the preceding expression.
{*n*,*m*}	Match at least *n* and at most *m* occurrences of the preceding expression.
^	Beginning of line.
$	End of line.
\b	Word boundary.
\B	Non-word boundary.
(*e*)	Capture expression *e*.
n	Back-reference to *n*th captured text.

Examples

The following regular expressions can be used with the editor's search-and-replace operations. To use the regular expression mode, the **Use regular expression** checkbox must be set in the search-and-replace dialog. Once enabled, regular expressions can be used in the **Find what** search string. The **Replace With** strings can use the "*n*" back-reference string to reference any captured strings.

"Find what"	**"Replace With"**	**Description**
u\w.d		Search for any-length string containing one or more word characters beginning with the character ‘u’ and ending in the character ‘d’.
^.*;$		Search for any lines ending in a semicolon.
(typedef.+\s+)(\S+);	\1TEST_\2;	Find C type definition and insert the string ‘TEST’ onto the beginning of the type name.

Locals window

The **Locals** window displays a list of all variables that are in scope of the selected stack frame in the **Call Stack**.

The **Locals** window has a toolbar and a main data display.

Button	Description
	Display the selected item in binary.
	Display the selected item in octal.
	Display the selected item in decimal.
	Display the selected item in hexadecimal.
	Display the selected item as a signed decimal.
	Display the selected item as a character or Unicode character.
	Set the range displayed in the active **Memory** window to span the memory allocated to the selected item.
	Sort variables alphabetically by name.
	Sort variables numerically by address or register number (default).

Using the Locals window

The **Locals** window shows the local variables of the active function when the debugger is stopped. The contents of the **Locals** window changes when you use the **Debug Location** toolbar items or select a new frame in the **Call Stack** window. When the program stops at a breakpoint, or is stepped, the **Locals** window updates to show the active stack frame. Items that have changed since they were previously displayed are highlighted in red.

To activate the Locals window:

- Choose **View** > **Locals** or press **Ctrl+Alt+L**.

When you select a variable in the main part of the display, the display-format button highlighted on the **Locals** window toolbar changes to show the selected item's display format.

To change the display format of a local variable:

- Right-click the item to change.
- From the shortcut menu, choose the desired display format.

—or—

- Click the item to change.
- On the **Locals** window toolbar, select the desired display format.

To modify the value of a local variable:

- Click the value of the local variable to modify.
- Enter the new value for the local variable. Prefix hexadecimal numbers with **0x**, binary numbers with **0b**, and octal numbers with **0**.

—or—

- Right-click the value of the local variable to modify.
- From the shortcut menu, select one of the commands to modify the local variable's value.

Globals window

The **Globals** window displays a list of all variables that are global to the program. The operations available on the entries in this window are the same as the **Watch** window, except you cannot add or delete variables from the **Globals** window.

Globals window user interface

The **Globals** window consists of a toolbar and main data display.

Globals toolbar

Button	Description
	Display the selected item in binary.
	Display the selected item in octal.
	Display the selected item in decimal.
	Display the selected item in hexadecimal.
	Display the selected item as a signed decimal.
	Display the selected item as a character or Unicode character.
	Set the range displayed in the active **Memory** window to span the memory allocated to the selected item.
	Sort variables alphabetically by name.
	Sort variables numerically by address or register number (default).

Using the Globals window

The **Globals** window shows the global variables of the application when the debugger is stopped. When the program stops at a breakpoint, or is stepped, the **Globals** window updates to show the active stack frame and new variable values. Items that have changed since they were previously displayed are highlighted in red.

To activate the Globals window:

- Choose **View** > **Globals** or press **Ctrl**+**Alt**+**G**.

Changing the display format

When you select a variable in the main part of the display, the display-format button highlighted on the **Globals** window toolbar changes to show the item's display format.

To change the display format of a global variable:

- Right-click the item to change.
- From the shortcut menu, choose the desired display format.

—or—

- Click the item to change.
- On the **Globals** window toolbar, select the desired display format.

To modify the value of a global variable:

- Click the value of the global variable to modify.
- Enter the new value for the global variable. Prefix hexadecimal numbers with **0x**, binary numbers with **0b**, and octal numbers with **0**.

Watch window

The **Watch** window provides a means to evaluate expressions and to display the results of those expressions. Typically, expressions are just the name of a variable to be displayed, but they can be considerably more complex; see **Debug expressions**. *Note:* expressions are always evaluated when your program stops, so the expression you are watching is the one that is in scope of the stopped program position.

The **Watch** window is divided into a toolbar and the main data display.

Button	Description
	Display the selected item in binary.
	Display the selected item in octal.
	Display the selected item in decimal.
	Display the selected item in hexadecimal.
	Display the selected item as a signed decimal.
	Display the selected item as a character or Unicode character.
	Set the range displayed in the active **Memory** window to span the memory allocated to the selected item.
	Remove the selected watch item.
	Remove all the watches.

Right-clicking a watch item shows a shortcut menu with commands that are not available from the toolbar.

Button	Description
	View pointer or array as a null-terminated string.
	View pointer or array as an array.
	View pointer value.
	Set watch value to zero.
	Set watch value to one.
	Increment watched variable by one.
	Decrement watched variable by one.

	Negated watched variable.
	Invert watched variable.
	View the properties of the watch value.

You can view details of the watched item using the **Properties** dialog.

Filename
The filename context of the watch item.

Line number
The line number context of the watch item.

(Name)
The name of the watch item.

Address
The address or register of the watch item.

Expression
The debug expression of the watch item.

Previous Value
The previous watch value.

Size In Bytes
The size of the watch item in bytes.

Type
The type of the watch item.

Value
The value of the watch item.

Using the Watch window

Each expression appears as a row in the display. Each row contains the expression and its value. If the value of an expression is structured (for example, an array), you can open the structure to see its contents.

The display updates each time the debugger locates to source code. So it will update each time your program stops on a breakpoint, or single steps, and whenever you traverse the call stack. Items that have changed since they were previously displayed are highlighted in red.

To activate the Watch window:

- Choose **View** > **Watch** > **Watch 1** or press **Ctrl+T, W, 1**.

You can show other **Watch** windows similarly.

You can add a new expression to be watched by clicking and typing into the last entry in the **Watch** window. You can change an expression by clicking its entry and editing its contents.

When you select a variable in the main part of the display, the display format button highlighted on the **Watch** window toolbar changes to show the item's display format.

To change the display format of an expression:

- Right-click the item to change.
- From the shortcut menu, choose the desired display format.

—or—

- Click the item to change.
- On the **Watch** window toolbar, select the desired display format.

The selected display format will then be used for all subsequent displays and will be preserved after the debug session stops.

For C programs, the interpretation of pointer types can be changed by right-clicking and selecting from the shortcut menu. A pointer can be interpreted as:

- a null-terminated ASCII string
- an array
- an integer
- dereferenced

To modify the value of an expression:

- Click the value of the local variable to modify.
- Enter the new value of the local variable. Prefix hexadecimal numbers with **0x**, binary numbers with **0b**, and octal numbers with **0**.

—or—

- Right-click the value of the local variable to modify.
- From the shortcut menu, choose one of the commands to modify the variable's value.

Register window

The **Register** windows show the values of both CPU registers and the processor's special function or peripheral registers. Because microcontrollers are becoming very highly integrated, it's not unusual for them to have hundreds of special function registers or peripheral registers, so SEGGER Embedded Studio provides four register windows. You can configure each register window to display one or more register groups for the processor being debugged.

A **Register** window has a toolbar and a main data display.

Button	Description
	Display the CPU, special function register, and peripheral register groups.
	Display the CPU registers.
	Hide the CPU registers.
	Force-read a register, ignoring the access attribute of the register.
	Update the selected register group.
	Set the active memory window to the address and size of the selected register group.

Using the registers window

Both CPU registers and special function registers are shown in the main part of the **Registers** window. When the program stops at a breakpoint, or is stepped, the **Registers** windows update to show the current values of the registers. Items that have changed since they were previously displayed are highlighted in red.

To activate the first register window:

- Choose **View** > **Registers** > **Registers 1** or press **Ctrl+T, R, 1**.

Other register windows can be similarly activated.

Displaying CPU registers

The values of the CPU registers displayed in the **Registers** window depend up upon the selected context. The selected context can be:

- The register state the CPU stopped in.
- The register state when a function call occurred using the Call Stack window.

- The register state of the currently selected thread using the the **Threads** window.
- The register state you supplied with the **Debug** > **Locate** operation.

To display a group of CPU registers:

- On the **Registers** window toolbar, click the **Groups** button.
- From the pop-up menu, select the register groups to display and deselect the ones to hide.

You can deselect all CPU register groups to allow more space in the display for special function registers or peripheral registers. So, for instance, you can have one register window showing the CPU registers and other register windows showing different peripheral registers.

Displaying special function or peripheral registers

The **Registers** window shows the set of register groups defined in the memory-map file the application was built with. If there is no memory-map file associated with a project, the **Registers** window will show only the CPU registers.

To display a special function or peripheral register:

- On the **Registers** toolbar, click the **Groups** button.
- From the pop-up menu, select the register groups to display and deselect the ones to hide.

Changing display format

When you select a register in the main part of the display, the display-format button highlighted on the **Registers** window toolbar changes to show the item's display format.

To change the display format of a register:

- Right-click the item to change.
- From the shortcut menu, choose the desired display format.

—or—

- Click the item to change.
- On the **Registers** window toolbar, select the desired display format.

Modifying register values

To modify the value of a register:

- Click the value of the register to modify.

- Enter the new value for the register. Prefix hexadecimal numbers with **0x**, binary numbers with **0b**, and octal numbers with **0**.

—or—

- Right-click the value of the register to modify.
- From the shortcut menu, choose one of the commands to modify the register value.

Modifying the saved register value of a function or thread may not be supported.

Memory window

The **Memory** window shows the contents of the connected target's memory areas and allows the memory to be edited. SEGGER Embedded Studio provides four memory windows, you can configure each memory window to display different memory ranges.

The **Memory** window has a toolbar and a data display/edit area

Field/Button	Description
Address	Address to display. This can be a numeric value or a debug expression.
Size	Number of bytes to display. This can be a number or a debug expression. If unspecified, the number of bytes required to fill the window will be automatically calculated.
Columns	Number of columns to display. If unspecified, the number of columns required to fill the window will be automatically calculated.
	Select binary display.
	Select octal display.
	Select unsigned decimal display.
	Select signed decimal display.
	Select hexadecimal display *(default)*.
	Select byte display *(default)*.
	Select 2-byte display.
	Select 4-byte display.
	Display both data and text *(default)*.
	Display data only.
	Display text only.
	Display an incrementing address range that starts from the selected address *(default)*.
	Display a decrementing address range that starts from the selected address.

	Display an incrementing address range that ends at the selected address.
	Display a decrementing address range that ends at the selected address.
	Evaluate the address and size expressions, and update the **Memory** window.

Using the memory window

The memory window does not show the complete address space of the target, instead you must enter both the address and the number of bytes to display. You can specify the address and size using numeric values or **debug expressions** which enable you to position the memory display at the address of a variable or at the value of a register. You can also specify whether you want the expressions to be evaluated each time the memory window is updated, or you can re-evaluate them yourself with the press of a button. Memory windows update each time your program stops on a breakpoint, after a single step and whenever you traverse the call stack. If any values that were previously displayed have changed, they are highlighted in red.

To activate the first Memory window:

- Choose **View** > **Memory** > **Memory 1** or press **Ctrl+T, M, 1**.

Other register windows can be similarly activated.

Using the mouse

You can move the memory window's edit cursor by clicking on a data or text entry.

The vertical scroll bar can be used to modify the address being viewed by clicking the up and down buttons, the page up and down areas or using the vertical scroll wheel when the scroll bar is at it's furthest extent. Holding down the **Shift** key while scrolling will prevent the address being modified.

Using the keyboard

Keystroke	Description
Up	Move the cursor up one line, or if the cursor is on the first line, move the address up one line.
Down	Move the cursor down one line, or if the cursor is on the last line, move the address down line line.
Left	Move the cursor left one character.
Right	Move the cursor right one character.
Home	Move the cursor to the first entry.
End	Move the cursor to the last entry.

PageUp	Move the cursor up one page, or if the cursor is on first page, move the address up one page.
PageDown	Move the cursor down one page, or if the cursor is on the last page, move the address down one page.
Ctrl+E	Toggle the cursor between data and text editing.

Editing memory

To edit memory, simply move the cursor to the data or text entry you want to modify and start typing. The memory entry will be written and read back as you type.

Shortcut menu commands

The shortcut menu contains the following commands:

Action	Description
Access Memory By Display Width	Access memory in terms of the display width.
Address Order	Specify whether the address range shown uses **Address** as the start or end address and whether addresses should increment or decrement.
Auto Evaluate	Re-evaluate **Address** and **Size** each time the **Memory** window is updated.
Auto Refresh	Specify how frequently the memory window should automatically refresh.
Export To Binary Editor	Create a binary editor with the current **Memory** window contents.
Save As	Save the current **Memory** window contents to a file. Supported file formats are **Binary File**, **Motorola S-Record File**, **Intel Hex File**, **TI Hex File**, and **Hex File**.
Load From	Load the current **Memory** window from a file. Supported file formats are **Binary File**, **Motorola S-Record File**, **Intel Hex File**, **TI Hex File**, and **Hex File**.

Display formats

You can set the **Memory** window to display 8-bit, 16-bit, and 32-bit values that are formatted as hexadecimal, decimal, unsigned decimal, octal, or binary. You can also specify how many columns to display.

Saving memory contents

You can save the displayed contents of the memory window to a file in various formats. Alternatively, you can export the contents to a binary editor to work on them.

You can save the displayed memory values as a binary file, Motorola S-record file, Intel hex file, or a Texas Instruments TXT file.

To save the current state of memory to a file:

- Select the start address and number of bytes to save by editing the **Start Address** and **Size** fields in the **Memory** window toolbar.
- Right-click the main memory display.
- From the shortcut menu, select **Save As**, then choose the format from the submenu.

To export the current state of memory to a binary editor:

- Select the start address and number of bytes to save by editing the **Start Address** and **Size** fields in the **Memory** window toolbar.
- Right-click the main memory display.
- Choose **Export to Binary Editor** from the shortcut menu.

Note that subsequent modifications in the binary editor will not modify memory in the target.

Copying to clipboard

You can copy the contents of the memory window to the clipboard as text. If an address range is selected, the data or text of the selected range will be copied to the clipboard depending on whether the selection has been made in the data or text view. If no address range is selected, the current memory window view will be copied to the clipboard.

Breakpoints window

The **Breakpoints** window manages the list of currently set breakpoints on the solution. Using the **Breakpoints** window, you can:

- Enable, disable, and delete existing breakpoints.
- Add new breakpoints.
- Show the status of existing breakpoints.

Breakpoints are stored in the session file, so they will be remembered each time you work on a particular project. When running in the debugger, you can set breakpoints on assembly code addresses. These low-level breakpoints appear in the **Breakpoints** window for the duration of the debug run but are not saved when you stop debugging.

When a breakpoint is reached, the matching breakpoint is highlighted in the **Breakpoints** window.

Breakpoints window layout

The **Breakpoints** window has a toolbar and a main breakpoint display.

Button	Description
	Create a new breakpoint using the **New Breakpoint** dialog.
	Toggle the selected breakpoint between enabled and disabled states.
	Remove the selected breakpoint.
	Move the insertion point to the statement where the selected breakpoint is set.
	Delete all breakpoints.
	Disable all breakpoints.
	Enable all breakpoints.
	Create a new breakpoint group and makes it active.

The main part of the **Breakpoints** window shows what breakpoints are set and the state they are in. You can organize breakpoints into folders, called *breakpoint groups*.

SEGGER Embedded Studio displays these icons to the left of each breakpoint:

Icon	Description

	**Enabled breakpoint** An enabled breakpoint will stop your program running when the breakpoint condition is met.
	**Disabled breakpoint** A disabled breakpoint will not stop the program when execution passes through it.
	**Invalid breakpoint** An invalid breakpoint is one where the breakpoint cannot be set; for example, no executable code is associated with the source code line where the breakpoint is set or the processor does not have enough hardware breakpoints.

Showing the Breakpoints window

To activate the Breakpoints window:

- Choose **Breakpoints** > **Breakpoints** or press **Ctrl+Alt+B**.

Managing single breakpoints

You can manage breakpoints in the **Breakpoint** window.

To delete a breakpoint:

- In the **Breakpoints** window, click the breakpoint to delete.
- From the **Breakpoints** window toolbar, click the **Delete** Breakpoint} button.

To edit a breakpoint:

- In the **Breakpoints** window, right-click the breakpoint to edit.
- Choose **Edit Breakpoint** from the shortcut menu.
- Edit the breakpoint in the **New Breakpoint** dialog.

- To toggle the enabled state of a breakpoint:

- In the **Breakpoints** window, right-click the breakpoint to enable or disable.
- Choose **Enable/Disable Breakpoint** from the shortcut menu.

—or—

- In the **Breakpoints** window, click the breakpoint to enable or disable.
- Press **Ctrl+F9**.

Breakpoint groups

Breakpoints are divided into *breakpoint groups*. You can use breakpoint groups to specify sets of breakpoints that are applicable to a particular project in the solution or for a particular debug scenario. Initially, there is a single breakpoint group, named *Default*, to which all new breakpoints are added.

To create a new breakpoint group:

- From the **Breakpoints** window toolbar, click the **New Breakpoint Group** button.

—or—

- From the **Debug** menu, choose **Breakpoints** then **New Breakpoint Group**.

—or—

- Right-click anywhere in the **Breakpoints** window.
- Choose **New Breakpoint Group** from the shortcut menu.

In the **New Breakpoint Group** dialog, enter the name of the breakpoint group.

When you create a breakpoint, it is added to the active breakpoint group.

To make a group the active group:

- In the **Breakpoints** window, right-click the breakpoint group to make active.
- Choose **Set as Active Group** from the shortcut menu.

To delete a breakpoint group:

- In the **Breakpoints** window, right-click the breakpoint group to delete.
- Choose **Delete Breakpoint Group** from the shortcut menu.

You can enable all breakpoints within a group at once.

To enable all breakpoints in a group:

- In the **Breakpoints** window, right-click the breakpoint group to enable.
- Choose **Enable Breakpoint Group** from the shortcut menu.

You can disable all breakpoints within a group at once.

To disable all breakpoints in a group:

- In the **Breakpoints** window, right-click the breakpoint group to disable.
- Choose **Disable Breakpoint Group** from the shortcut menu.

Managing all breakpoints

You can delete, enable, or disable all breakpoints at once.

To delete all breakpoints:

- Choose **Breakpoints** > **Clear All Breakpoints** or press **Ctrl+Shift+F9**.

—or—

- On the **Breakpoints** window toolbar, click the **Delete All Breakpoints** button.

To enable all breakpoints:

- Choose **Breakpoints** > **Enable All Breakpoints** or press **Ctrl+B, N**.

—or—

- On the **Breakpoints** window toolbar, click the **Enable All Breakpoints** button.

To disable all breakpoints:

- Choose **Breakpoints** > **Disable All Breakpoints** or press **Ctrl+B, X**.

—or—

- On the **Breakpoints** window toolbar, click the **Disable All Breakpoints** button.

Call Stack window

The **Call Stack** window displays the list of function calls (stack frames) that were active when program execution halted. When execution halts, SEGGER Embedded Studio populates the call-stack window from the active (currently executing) task. For simple, single-threaded applications not using the SEGGER Embedded Studio tasking library, there is only a single task; but for multi-tasking programs that use the SEGGER Embedded Studio Tasking Library, there may be any number of tasks. SEGGER Embedded Studio updates the **Call Stack** window when you change the active task in the **Threads** window.

The **Call Stack** window has a toolbar and a main call-stack display.

Button	Description
	Move the insertion point to where the call was made to the selected frame.
	Set the debugger context to the selected stack frame.
	Move the debugger context down one stack to the called function.
	Move the debugger context up one stack to the calling function.
	Select the fields to display for each entry in the call stack.
	Set the debugger context to the most recent stack frame and move the insertion point to the currently executing statement.

The main part of the **Call Stack** window displays each unfinished function call (active stack frame) at the point when program execution halted. The most recent stack frame is displayed at the bottom of the list and the oldest is displayed at the top of the list.

SEGGER Embedded Studio displays these icons to the left of each function name:

Icon	Description
	Indicates the stack frame of the current task.
	Indicates the stack frame selected for the debugger context.
	Indicates that a breakpoint is active and when the function returns to its caller.

These icons can be overlaid to show, for instance, the debugger context and a breakpoint on the same stack frame.

Showing the call-stack window

To activate the Call Stack window:

- Choose **View** > **Call Stack** or press **Ctrl+Alt+S**.

Configuring the call-stack window

Each entry in the **Call Stack** window displays the function name and, additionally, parameter names, types, and values. You can configure the **Call Stack** window to show varying amounts of information for each stack frame. By default, SEGGER Embedded Studio displays all information.

To show or hide a field:

1. On the **Call Stack** toolbar, click the **Options** button on the far right.
2. Select the fields to show, and deselect the ones that should be hidden.

Changing the debugger context

You can select the stack frame for the debugger context from the **Call Stack** window.

To move the debugger context to a specific stack frame:

- In the **Call Stack** window, double-click the stack frame to move to.

—or—

- In the **Call Stack** window, select the stack frame to move to.
- On the **Call Stack** window's toolbar, click the **Switch To Frame** button.

—or—

- In the **Call Stack** window, right-click the stack frame to move to.
- Choose **Switch To Frame** from the shortcut menu.

The debugger moves the insertion point to the statement where the call was made. If there is no debug information for the statement at the call location, SEGGER Embedded Studio opens a disassembly window at the instruction.

To move the debugger context up one stack frame:

- On the **Call Stack** window's toolbar, click the **Up One Stack Frame** button.

—or—

- On the **Debug Location** toolbar, click the **Up One Stack Frame** button.

—or—

- Press **Alt+-**.

The debugger moves the insertion point to the statement where the call was made. If there is no debug information for the statement at the call location, SEGGER Embedded Studio opens a disassembly window at the instruction.

To move the debugger context down one stack frame:

- On the **Call Stack** window's toolbar, click the **Down One Stack Frame** button.

—or—

- On the **Debug Location** toolbar, click the **Down One Stack Frame** button.

—or—

- Press **Alt++**.

The debugger moves the insertion point to the statement where the call was made. If there is no debug information for the statement at the call location, SEGGER Embedded Studio opens a disassembly window at the instruction.

Setting a breakpoint on a return to a function

To set a breakpoint on return to a function:

- In the **Call Stack** window, click the stack frame on the function to stop at on return.
- On the **Build** toolbar, click the **Toggle Breakpoint** button.

—or—

- In the **Call Stack** window, click the stack frame on the function to stop at on return.
- Press **F9**.

—or—

- In the **Call Stack** window, right-click the function to stop at on return.
- Choose **Toggle Breakpoint** from the shortcut menu.

Threads window

The **Threads** window displays the set of executing contexts on the target processor structured as a set of queues.

To activate the Threads window:

- Choose **View** > **More Debug Windows** > **Threads** or press **Ctrl**+**Alt**+**H**.

The window is populated using the threads script, which is a JavaScript program store in a file whose file-type project option is "Threads Script" (or is called `threads.js`) and is in the project that is being debugged.

When debugging starts, the threads script is loaded and the **function init()** is called to determine which columns are displayed in the **Threads** window.

When the application stops on a breakpoint, the function **update()** is called to create entries in the **Threads** window corresponding to the columns that have been created together with the saved execution context (register state) of the thread. By double-clicking one of the entries, the debugger displays its saved execution context—to put the debugger back into the default execution context, use **Show Next Statement**.

Writing the threads script

The threads script controls the **Threads** window with the **Threads** object.

The methods **Threads.setColumns**, **Threads.setSortByNumber** and **Threads.setColor** can be called from the **function init()**.

```
function init()
{
  Threads.setColumns("Name", "Priority", "State", "Time");
  Threads.setSortByNumber("Time");
  Threads.setColor("State", "Ready", "Executing", "Waiting");
}
```

The above example creates the named columns **Name**, **Priority**, **State**, and **Time** in the **Threads** window, with the **Time** column sorted numerically rather than alphabetically. The states **Ready**, **Executing** and **Waiting** will have yellow, green and red colored pixmaps respectively.

If you don't supply the **function init()** in the threads script, the **Threads** window will create the default columns **Name**, **Priority**, and **State**.

The methods **Threads.clear()**, **Threads.newqueue()**, and **Threads.add()** can be called from the **function update()**.

The **Threads.clear()** method clears the **Threads** window.

The **Threads.newqueue()** function takes a string argument and creates a new, top-level entry in the **Threads** window. Subsequent entries added to this window will go under this entry. If you don't call this, new entries will all be at the top level of the **Threads** window.

The **Threads.add()** function takes a variable number of string arguments, which should correspond to the number of columns displayed by the **Threads** window. The last argument to the **Threads.add()** function should be an array (possibly empty) containing the registers of the thread or, alternatively, a handle that can be supplied a call to the threads script **function getregs(handle)**, which will return an array when the thread is selected in the **Threads** window. The array containing the registers should have elements in the same order in which they are displayed in the CPU **Registers** display—typically this will be in register-number order, e.g., **r0**, **r1**, and so on.

```
function update()
{
  Threads.clear();
  Threads.newqueue("My Tasks");
  Threads.add("Task1", "0", "Executing", "1000", [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]);
  Threads.add("Task2", "1", "Waiting", "2000", [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]);
}
```

The above example will create a fixed output on the **Threads** window and is here to demonstrate how to call the methods.

To get real thread state, you need to access the debugger from the threads script. To do this, you can use the JavaScript method **Debug.evaluate("expression")**, which will evaluate the string argument as a debug expression and return the result. The returned result will be an object if you evaluate an expression that denotes a structure or an array. If the expression denotes a structure, each field can be accessed by using its field name.

So, if you have structs in the application as follows…

```
struct task {
  char *name;
  unsigned char priority;
  char *state;
  unsigned time;
  struct task *next;
  unsigned registers[17];
  unsigned thread_local_storage[4];
};

struct task task2 =
{
  "Task2",
   1,
   "Waiting",
   2000,
   0,
   { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 },
   { 0,1,2,3 }
};

struct task task1 =
{
  "Task1",
  0,
  "Executing",
  1000,
  &task2,
  { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 },
```

```
  { 0,1,2,3 }
};
```

…you can **update()** the **Threads** window using the following:

```
task1 = Debug.evaluate("task1");
Threads.add(task1.name, task1.priority, task1.state, task1.time, task1.registers);
```

You can use pointers and C-style cast to enable linked-list traversal.

```
var next = Debug.evaluate("&task1");
while (next)
  {
    var xt = Debug.evaluate("*(struct task*)"+next);
    Threads.add(xt.name, xt.priority, xt.state, xt.time, xt.registers);
    next = xt.next;
  }
```

Note that, if the threads script goes into an endless loop, the debugger—and consequently SEGGER Embedded Studio—will become unresponsive and you will need to kill SEGGER Embedded Studio using a task manager. Therefore, the above loop is better coded as follows:

```
var next = Debug.evaluate("&task1");
var count = 0;
while (next && count < 10)
  {
    var xt = Debug.evaluate("*(struct task*)"+next);
    Threads.add(xt.name, xt.priority, xt.state, xt.time, xt.registers);
    next = xt.next;
    count++;
  }
```

You can speed up the **Threads** window update by not supplying the registers of the thread to the **Threads.add()** function. To do this, you should supply a handle/pointer to the thread as the last argument to the **Threads.add()** function. For example:

```
var next = Debug.evaluate("&task1");
var count = 0;
while (next && count < 10)
  {
    var xt = Debug.evaluate("*(struct task*)"+next);
    Threads.add(xt.name, xt.priority, xt.state, xt.time, next);
    next=xt.next;
    count++;
  }
```

When the thread is selected, the **Threads** window will call **getregs(x)** in the threads script. That function should return the array of registers, for example:

```
function getregs(x)
{
  return Debug.evaluate("((struct task*)"+x+")->registers");
}
```

If you use thread local storage, implementing the **gettls(x)** function enables you to return the base address of the thread local storage, for example:

```
function gettls(x)
{
  return Debug.evaluate("((struct task*)"+x+")->thread_local_storage");
}
```

The **gettls(x)** function can also be called with null as a parameter. In this case you will have to evaluate an expression that returns the current thread local storage, for example:

```
function gettls(x)
{
  if (x==null)
    x = Debug.evaluate("&currentTask");
  return Debug.evaluate("((struct task*)"+x+")->thread_local_storage");
}
```

The debugger may require the name of a thread which you can provide by implementing the **getname(x)** function, for example:

```
function getname(x)
{
  return Debug.evaluate("((struct task*)"+x+")->name");
}
```

Adding extra queues to the threads window

You can add extra information to the threads window to display other RTOS queues. In the **function init()** you can use **Threads.setColumns2** to create an additional display in the threads window, for example:

```
function init()
{
  ...
  Threads.setColumns2("Timers", "Id(Timers)", "Name", "Hook", "Timeout", "Period", "Active");
```

The first argument is identifier of the queue which is also supplied to **Threads.add2** in the **function update()** as follows

```
function update()
{
  ...
  Threads.add2("Timers", "0x1FF0A30", "MyTimer", "0x46C8 (Timer50)", "50(550)", "50", "1");
```

Execution Profile window

The **Execution Profile** window shows a list of source locations and the number of times those source locations have been executed. This window is only available for targets that support the collection of jump trace information.

To activate the Execution Profile window:

- Choose **View** > **More Debug Windows** > **Execution Profile** or press **Ctrl**+**T, P**.

The count value displayed is the number of times the first instruction of the source code location has been executed. The source locations displayed are target dependent: they could represent each statement of the program or each jump target of the program. If however the debugger is in intermixed or disassembly mode then the count values will be displayed on a per instruction basis.

The execution counts window is updated each time your program stops and the window is visible so if you have this window displayed then single stepping may be slower than usual.

Execution Trace window

The trace window displays historical information on the instructions executed by the target.

To activate the Trace window:

- Choose **View** > **More Debug Windows** > **Execution Trace** or press **Ctrl**+**T, T**.

The type and number of the trace entries depends upon the target that is connected when gathering trace information. Some targets may trace all instructions, others may trace jump instructions, and some may trace modifications to variables. You'll find the trace capabilities of your target on the shortcut menu.

Each entry in the trace window has a unique number, and the lower the number the earlier the trace. You can click on the header to show earliest to latest or the latest to earliest trace entries. If a trace entry can have source code located to it then double-clicking the trace entry will show the appropriate source display.

Some targets may provide timing information which will be displayed in the ticks column.

The trace window is updated each time the debugger stops when it is visible so single stepping is likely to be slower if you have this window displayed.

Debug file search editor

When a program is built with debugging enabled, the debugging information contains the paths and filenames of all the source files for the program in order to allow the debugger to find them. If a program or library linked into the program is on a different machine than the one on which it was compiled, or if the source files were moved after the program was compiled, the debugger will not be able to find the source files.

In this situation, the simplest way to help SEGGER Embedded Studio find the source files is to add the directory containing the source files to one of its source-file search paths. Alternatively, if SEGGER Embedded Studio cannot find a source file, it will prompt you for its location and will record its new location in the source-file map.

Debug source-file search paths

Debug's source-file search paths can be used to help the debugger locate source files that are no longer located where they were at compile time. When a source file cannot be found, the search-path directories will be checked, in turn, to see if they contain the source file. SEGGER Embedded Studio maintains two debug source-file search paths:

- *Project-session search path*: This path is for the current project session and does not apply to all projects.
- *The global search path*: This system-wide path applies to all projects.

The project-session search path is checked before the global search path.

To edit the debug search paths:

- Choose **Debug** > **Options** > **Search Paths**.

Debug source file map

If a source file cannot be found while debugging and the debugger has to prompt the user for its location, the results are stored in the debug source file map. The debug source file map simply correlates, or *maps*, the original pathnames to the new locations. When a file cannot be found at its original location or in the debug search paths, the debug source file map is checked to see if a new location has been recorded for the file or if the user has specified that the file does not exist. Each project session maintains its own source file map, the map is not shared by all projects.

To view the debug source file map:

- Choose **Debug** > **Options** > **Search Paths**.

To remove individual entries from the debug source file map:

- Choose **Debug** > **Options** > **Search Paths**.

- Right-click the mapping to delete.
- Choose **Delete Mapping** from the shortcut menu.

To remove all entries from the debug source file map:

- Choose **Debug** > **Options** > **Search Paths**.
- Right-click any mapping.
- Choose **Delete All Mappings** from the shortcut menu.

Debug Terminal window

The **Debug Terminal** window displays debug output from the target application and can also be used to be provide debug input to the target application.

To activate the Debug Terminal window:

- Choose **View** > **Debug Terminal**.

Breakpoint expressions

The debugger can set breakpoints by evaluating simple C-like expressions. Note that the exact capabilities offered by the hardware to assist in data breakpointing will vary from target to target; please refer to the particular target interface you are using and the capabilities of your target silicon for exact details. The simplest expression supported is a symbol name. If the symbol name is a function, a breakpoint occurs when the first instruction of the symbol is about to be executed. If the symbol name is a variable, a breakpoint occurs when the symbol has been accessed; this is termed a *data breakpoint*. For example, the expression `x` will breakpoint when *x* is accessed. You can use a debug expression (see **Debug expressions**) as a breakpoint expression. For example, `x[4]` will breakpoint when element 4 of array *x* is accessed, and `@sp` will breakpoint when the *sp* register is accessed.

Data breakpoints can be specified, using the == operator, to occur when a symbol is accessed with a specific value. The expression `x == 4` will breakpoint when *x* is accessed and its value is 4. The operators <, >=, >;, >=, ==, and != can be used similarly. For example, `@sp <= 0x1000` will breakpoint when register **sp** is accessed and its value is less than or equal to 0x1000.

You can use the operator '&' to mask the value you wish to break on. For example, `(x & 1) == 1` will breakpoint when *x* is accessed and has an odd value.

You can use the operator '&&' to combine comparisons. For example…

```
(x >= 2) && (x <= 14)
```

…will breakpoint when *x* is accessed and its value is between 2 and 14.

You can specify an arbitrary memory range using an array cast expression. For example, `(char[256])(0x1000)` will breakpoint when the memory region 0x1000–0x10FF is accessed.

You can specify an inverse memory range using the **!** operator. For example `!(char[256])(0x1000)` will breakpoint when memory outside the range 0x1000–0x10FF is accessed.

Debug expressions

The debugger can evaluate simple expressions that can be displayed in the **Watch** window or as a tool-tip in the code editor.

The simplest expression is an identifier the debugger tries to interpret in the following order:

- an identifier that exists in the scope of the current context.
- the name of a global identifier in the program of the current context.

Numbers can be used in expressions. Hexadecimal numbers must be prefixed with `0x`.

Registers can be referenced by prefixing the register name with `@`.

The standard C and C++ operators `!`, ~, `*`, `/`, %, +, `-`, >>, <<, <, <=, >, >=, ==, `|`, &, ^, &&, and `||` are supported on numeric types.

The standard assignment operators =, +=, `-=`, `*=`, `/=`, %=, >>, >>=, <<=, &=, `|=`, ^= are supported on numeric types.

The array subscript operator '[]' is supported on array and pointer types.

The structure access operator '.' is supported on structured types (this also works on pointers to structures), and -> works similarly.

The dereference operator (prefix '*') is supported on pointers, the address-of (prefix '&') and **sizeof** operators are supported.

The `addressof(`*filename*`,` *linenumber*`)` operator will return the address of the specified source code line number.

Function calling with parameters and return results.

Casting to basic pointer types is supported. For example, (unsigned char *)0x300 can be used to display the memory at a given location.

Casting to basic array types is supported. For example, (unsigned char[256])0x100 can be used to reference a memory region.

Arrays can be sliced using '[a:b]' where 'a' is the first element and 'b' is the last element to display.

Operators have the precedence and associativity one expects of a C-like programming language.

Terminal emulator window

The **Terminal Emulator** window contains a basic serial-terminal emulator that allows you to receive and transmit data over a serial interface.

To activate the Terminal Emulator window:

- Choose **Tools** > **Terminal Emulator** > **Terminal Emulator** or press **Ctrl+Alt+M**.

To use the terminal emulator:

1. Set the required terminal emulator properties.
2. Connect the terminal emulator to the communications port by clicking the button on the toolbar or by selecting **Connect** from the shortcut menu.

Once connected, any input in the **Terminal Emulator** window is sent to the communications port and any data received from the communications port is displayed on the terminal.

Connection may be refused if the communication port is in use by another application or if the port doesn't exist.

To disconnect the terminal emulator:

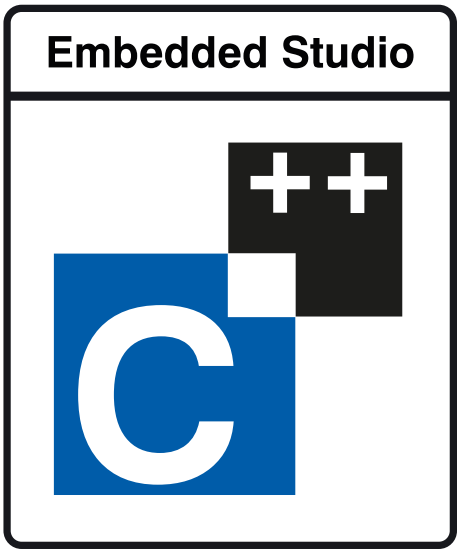
1. Disconnect the communications port by clicking the **Disconnect** icon on the toolbar or by right-clicking to select **Disconnect** from the shortcut menu.

This will release the communications port for use in other applications.

Supported control codes

The terminal supports a limited set of control codes:

Control code	Description
<BS>	Backspace
<CR>	Carriage return
<LF>	Linefeed
<ESC>[{attr1};...;{attrn}m	Set display attributes. The attributes 2-Dim, 5-Blink, 7-Reverse, and 8-Hidden are not supported.



Command-line options

This section describes the command-line options accepted by SEGGER Embedded Studio.

Usage

emStudio [*options*…] [*files*…]

-D (Define macro)

Syntax

-D *macro=value*

Description

Define a SEGGER Embedded Studio macro value.

-noclang (Disable Clang support)

Syntax

-noclang

Description

Disable Clang support.

-noload (Disable loading of last project)

Syntax

-noload

Description

Disable loading of last project on startup.

-packagesdir (Specify packages directory)

Syntax

-**packagesdir** *dir*

Description

Override the default value of the **$(PackagesDir)** macro.

-permit-multiple-studio-instances (Permit multiple studio instances)

Syntax

-permit-multiple-studio-instances

Description

Allow multiple instances of SEGGER Embedded Studio to run at the same time. This behaviour can also be enabled using the **Environment** > **Startup Options** > **Allow Multiple SEGGER Embedded Studios** environment option.

-rootuserdir (Set the root user data directory)

Syntax

-rootuserdir *dir*

Description

Set the SEGGER Embedded Studio root user data directory.

-save-settings-off (Disable saving of environment settings)

Syntax

-save-settings-off

Description

Disable the saving of modified environment settings.

-set-setting (Set environment setting)

Syntax

-set-setting *environment_setting=value*

Description

Sets an environment setting to a specified value. For example:

```
-set-setting "Environment/Build/Show Command Lines=Yes"
```

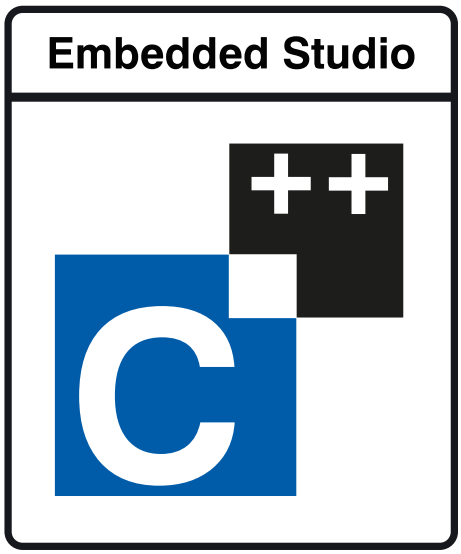
-templatesfile (Set project templates path)

Syntax

-templatesfile *path*

Description

Sets the search path for finding project template files.



Uninstalling SEGGER Embedded Studio

This section describes how to completely uninstall SEGGER Embedded Studio for each supported operating system:

- **Uninstalling SEGGER Embedded Studio from Windows**
- **Uninstalling SEGGER Embedded Studio from macOS**
- **Uninstalling SEGGER Embedded Studio from Linux**

Uninstalling SEGGER Embedded Studio from Windows

Removing user data and settings

The uninstaller does not remove any user data such as settings or installed packages. To completely remove the user data you will need to carry out the following operations for each user that has used SEGGER Embedded Studio on your system.

To remove user data using SEGGER Embedded Studio:

1. Start SEGGER Embedded Studio.
2. Click **Tools** > **Admin** > **Remove All User Data...**

Alternatively, if SEGGER Embedded Studio has already been uninstalled you can manually remove the user data as follows:

1. Click the Windows Start button.

2. Type *%LOCALAPPDATA%* in the search field and press enter to open the local application data folder.
3. Open the *SEGGER* folder.
4. Open the *SEGGER Embedded Studio* folder.
5. Delete the *v8* folder.
6. If you want to delete user data for all versions of the software, delete the *SEGGER Embedded Studio* folder as well.

Uninstalling SEGGER Embedded Studio

To uninstall SEGGER Embedded Studio:

1. If SEGGER Embedded Studio is running, click **File** > **Exit** to shut it down.
2. Click the Start Menu and select Control Panel. The Control Panel window will open.
3. In the Control Panel window, click the **Uninstall a program** link under the Programs section.
4. From the list of currently installed programs, select **SEGGER Embedded Studio 8.26b**.
5. To begin the uninstall, click the **Uninstall** button at the top of the list.

Uninstalling SEGGER Embedded Studio from macOS

Removing user data and settings

Uninstalling does not remove any user data such as settings or installed packages. To completely remove the user data you will need to carry out the following operations for each user that has used SEGGER Embedded Studio on your system.

To remove user data using SEGGER Embedded Studio:

1. Start SEGGER Embedded Studio.
2. Click **Tools** > **Admin** > **Remove All User Data...**

Alternatively, if SEGGER Embedded Studio has already been uninstalled you can manually remove the user data as follows:

1. Open Finder.
2. Go to the *$HOME/Library/SEGGER/SEGGER Embedded Studio* directory.
3. Drag the *v8* folder to the Trash.
4. If you want to delete user data for all versions of the software, drag the *SEGGER Embedded Studio* folder to the Trash as well.

Uninstalling SEGGER Embedded Studio

To uninstall SEGGER Embedded Studio:

1. If SEGGER Embedded Studio is running, shut it down.
2. Open the */Applications/SEGGER* folder in Finder.
3. Drag the *SEGGER Embedded Studio 8.26b* folder to the Trash.

Uninstalling SEGGER Embedded Studio from Linux

Removing user data and settings

The uninstaller does not remove any user data such as settings or installed packages. To completely remove the user data you will need to carry out the following operations for each user that has used SEGGER Embedded Studio on your system.

To remove user data using SEGGER Embedded Studio:

1. Start SEGGER Embedded Studio.
2. Click **Tools** > **Admin** > **Remove All User Data...**

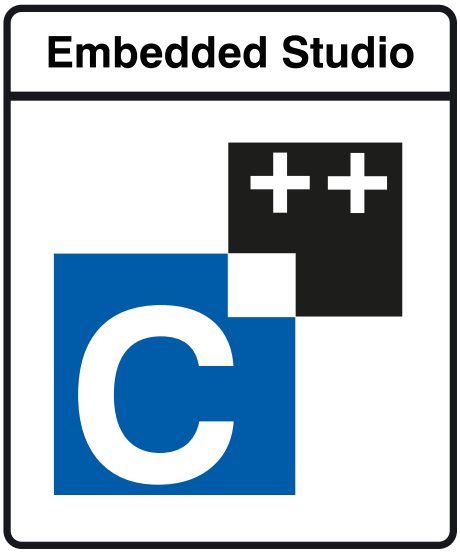
Alternatively, if SEGGER Embedded Studio has already been uninstalled you can manually remove the user data as follows:

1. Open a terminal window or file browser.
2. Go to the *$HOME/.segger/SEGGER Embedded Studio* directory.
3. Delete the *v8* directory.
4. If you want to delete user data for all versions of the software, delete the *SEGGER Embedded Studio* directory as well.

Uninstalling SEGGER Embedded Studio

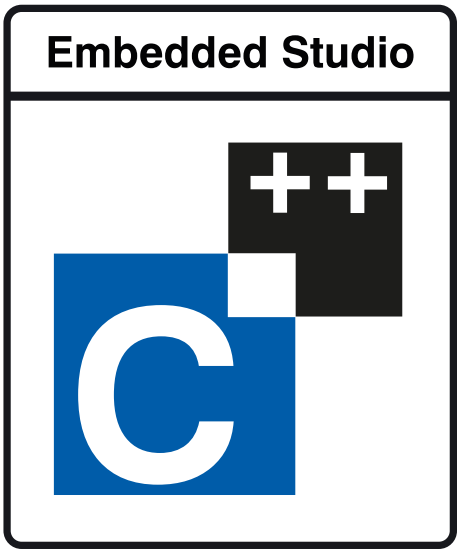
To uninstall SEGGER Embedded Studio:

1. If SEGGER Embedded Studio is running, click **File** > **Exit** to shut it down.
2. Open a terminal window.
3. Go to the SEGGER Embedded Studio bin directory (this is */opt/SEGGER/segger_embedded_studio_8.26b/bin* by default).
4. Run *sudo ./uninstall* to start the uninstaller.



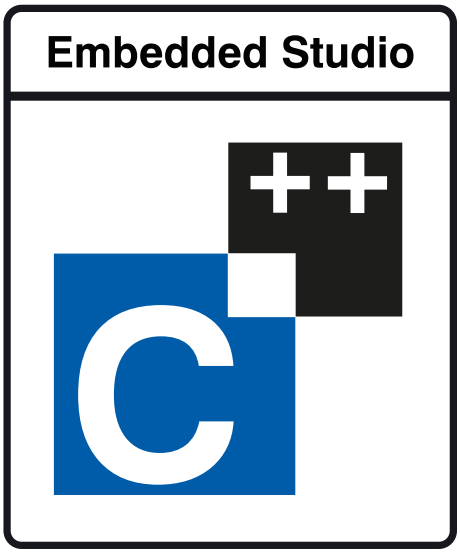
The SEGGER Compiler

You can use the SEGGER compiler by setting the project option **Compiler** to **SEGGER**.



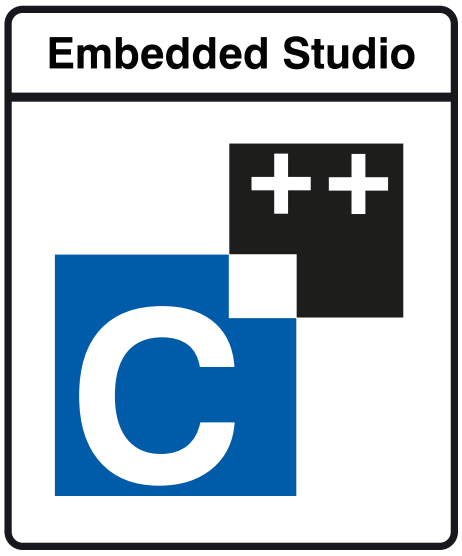
The SEGGER Linker

You can use the SEGGER linker by setting the project option **Linker** to **SEGGER**.



The SEGGER Runtime Library

The SEGGER runtime library is used by default.



Utilities Reference

Command-Line Compiler

This section describes the switches accepted by the compiler driver, **cc**. The compiler driver is capable of controlling compilation by all supported language compilers and the final link by the linker. It can also construct libraries automatically.

File naming conventions

The compiler driver uses file extensions to distinguish the language the source file is written in. The compiler driver recognizes the extension **.c** as C source files, **.s** and **.asm** as assembly code files.

The compiler driver recognizes the extension **.hzo** as object files, **.hza** as library files and **.xml** as special-purpose XML files.

We strongly recommend that you adopt these extensions for your source files and object files because you'll find that using the tools is much easier if you do.

C language files

When the compiler driver finds a file with a **.c** extension, it runs the C compiler to convert it to object code. Alternatively you can specify that it is a C file using

```
cc -x c cfile.notc ...
```

Assembly language files

When the compiler driver finds a file with a **.s** or **.asm** extension, it runs the C preprocessor and then the assembler to convert it to object code. Alternatively you can specify that it is an assembly language file using

```
cc -x asm asmfile.nots ...
```

Object code files

When the compiler driver finds a file with a **.hzo** or **.hza** extension, it passes it to the linker to include it in the final application.

Compilation

To compile or assemble a file you should supply the **-c** option together with the source file and provide a name for the output file using the **-o** option

```
cc -c file.c -o file.o
```

if you don't supply an output name then the output file will be use the basename of the source file.

You can supply the file to be compiled from the standard input using the

```
cc -c - -o main.o
int main()
```

```
{
  return 3;
}
<EOF>
```

You can preprocess the source file rather than compile it using the **-E** option

```
cc -E main.c
```

This will send the output to the standard output or you can use **-o** to send it to a named file.

You can show the preprocessor defines that are defined for the compilation using the **-dM** option

```
cc -E -dM main.c
```

You can supply preprocessor defines and include directories using **-D** and **-I** options

```
cc -c file.c -Dmydefine -Imyincludedir
```

You can include a file before compilation using **-include**

```
cc -c file.c -include file.h
```

There is also a variant that will just use the #defines that are declared in the included file

```
cc -c file.c -imacros file.h
```

If you wish to not use the default C/C++ library you can use **-I-** and then supply your own system library directory using **-isystem**

```
cc -c file.c -I- -isystemmysystemincludedir
```

You can using the **-g** option to include debugging information in the output file

```
cc -c file.c -g
```

You can use the **-O** option to set the desired optimization level

```
cc -c file.c -O0
```

Linking

You can compile/link a number of files with the standard libraries

```
cc Cortex_M_Startup.s thumb_crt0.s main.o -o main.elf
```

You'll also need to supply linker control details. There are a number of ways of doing this

```
cc .. -placement ram_placement.xml -placementsegments "SRAM RW 0x0 0x1000" -ereset_handler
```

```
cc .. -memorymap map.xml -placementsegments "SRAM RW 0x0 0x1000" -ereset_handler
```

```
cc .. -Tlinker.icf -ereset_handler
```

Target Selection

Advanced

You can create a precompile header using the **-pch** option

```
cc -c -xc -pch main.h -o main.h.pch
```

Note that the output file must be in the same directory as the input file. You can use this precompiled header file

```
cc -c main.c -include-pch main.h.pch
```

You can create a C++ 20 module using

```
cc++ -c -xc++ main.cxx -std=c++20 -fmodules-ts -fmodule-file=main=main.o -o main.o
```

The module file will be named either .gcm or .pcm and can be used by another file using

```
cc++ -c -xc++ another.cxx -std=c++20 -fmodules-ts -fmodule-file=main=main.o -o another.o
```

Options:

Option	Description
-	input is taken from standard input
-###	show commands but don't execute them
-allow-multiple-definition	allow multiple symbol definition when linking
-ansi	enforce ANSI checking
-ar	create library from input files
-arch=val	set cpu architecture to 'val', use list to display supported
-arm	generate ARM code
-arm64	generate ARM64 code
-be	big endian target
-be8	big endian target
-builtins	use builtin compiler functions
-c	compile the files, no link/library
-cmselib=l	create cmse output library in 'l'
-codec=c	set file codec to 'c', use list to display supported
-common	allocate global variables in the common section
-cpu=val	set cpu core to 'val', use list to display supported
-depend file	generate dependency file in 'file'
-dependu file	generate dependency file in 'file' with user header files only

-dM	show #defines
-Dname	define the preprocessor macro 'name'
-Dname=val	define the preprocessor macro 'name' as 'val'
-dname=val	define the linker symbol 'name' as 'val'
-E	preprocess file and write to standard output
-emit-relocs	emit relocations into executable
-ename	set program entry symbol to 'name'
-exceptions	enable C++ exceptions
-Fbin	create an additional binary output file
-fbuiltin	enable compiler builtin functions
-fcommon	place global variables in COMMON section
-fcoroutines	enable C++ coroutine support
-fdebug-types-section	generate .debug_types section
-fdiagnostics-color=always	color diagnostic output of the compiler
-fdiagnostics-color=never	do not color diagnostic output of the compiler
-fdiagnostics-show-caret	show caret in diagostic output of the compiler
-fexceptions	enable C++ exception support
-Fhex	create an additional hex output file
-fill=b	fill gaps in the additional output file with byte 'b'
-flto	generate code suitable for link time optimization
-fmath-errno	set errno after calling math functions
-fmodule-file='name'	get module dependencies from the file 'name'
-fmodules-ts	enable c++20 modules
-fno-builtin	disable compiler builtin functions
-fno-common	place global variables in bss section
-fno-diagnostics-show-caret	do not show caret in diagostic output of the compiler
-fno-exceptions	disable C++ exception support
-fno-math-errno	set errno after calling math functions
-fno-omit-frame-pointer	disable framepointer generation
-fno-rtti	disable C++ RTTI support
-fno-short-enums	enumerations are int sized
-fno-short-wchar	wide characters are 32-bit
-fno-signed-char	char is considered to be unsigned char
-fomit-frame-pointer	disable framepointer generation

-fpabi=hard	generate FPU instructions passing fp arguments in FPU registers
-fpabi=soft	do not generate FPU instructions
-fpabi=softfp	generate FPU instructions passing fp arguments in CPU registers
-fpu=val	set fpu to 'val', use list to display supported
-framepointer	generate code to maintain a frame pointer register
-frtti	enable C++ RTTI support
-fshort-enums	enumerations are minimal container sized
-fshort-wchar	wide characters are 16-bit
-fsigned-char	char is considered to be signed char
-Fsrec	create an additional srec output file
-ftree-vectorize	perform vectorization on trees
-funwind-tables	generate unwind tables
-g1	generate only backtrace and line number debugging information
-g2	generate level 1 and variable display debugging information
-g3	generate level 2 and macro display debugging information
-gcc	use gcc assembler/compiler/lto
-gcc-target=name	select gcc 'name' tools to use
-gdwarf-2	generate dwarf-2 debugging information
-gdwarf-3	generate dwarf-3 debugging information
-gdwarf-4	generate dwarf-4 debugging information
-gdwarf-5	generate dwarf-5 debugging information
-gnu-ld	use gnu linker
-gpubnames	generate .debug_pubnames and .debug_pubtypes sections
-hascmse	v8m architecture has cmse instructions
-hascrc	v8a architecuture has crc instructions
-hascrypto	v8a architecture has crypto instructions
-hasdsp	v8m architecture has dsp instructions
-hasidiv	v7ar architecture has integer divide instructions
-hassmallmultiplier	cortex-m0/m0+/m1 architecture has small multiplier
-heap=basic	use basic heap implementation
-heap=minimal	use minimal heap implementation

`-heap=rt`	use real-time heap implementation
`-help`	show this text
`-I-`	do not search any standard directories for include files
`-Idir`	add 'dir' to the end of the user include search list
`-imacros` file	same as -include but only keep #defines
`-include` file	#include 'file' before the source file
`-include-pch` file	#include precompiled header 'file' before the source file
`-inputfiles` file	list of files in 'file' to link or archive
`-instrument`	instrument functions
`-io=none`	use user supplied implementation for library io
`-io=rtt`	use RTT implementation for library io
`-io=semihost`	use semihosted implementation for library io
`-io=semihostf`	use host formatted semihosted implementation for library io
`-io=swo`	use SWO implementation for library io
`-io=swoi`	use SWO with interrupts disabled implementation for library io
`-isystem` dir	add 'dir' to the end of the system include search list
`-Jdir`	add 'dir' to the end of the system include search list
`-kasm`	keep assembly code output
`-kind`	keep indirect files
`-kldscript`	keep generated linker script
`-klto`	keep lto generated files
`-Kname`	keep symbol 'name' in the linked output
`-kpp`	keep preprocessor output
`-l-`	disable linking of standard libraries
`-Ldir`	search directory 'dir' to find libraries
`-le`	little endian target
`-libdir` dir	specify system library directory 'dir'
`-lname`	search library 'name' to resolve symbols
`-locales=minimal`	locales implementation
`-longcalls`	generate long calling instruction sequences
`-lunwind`	generate stack unwind tables
`-M`	generate linkage map file

`-march=val`	set cpu architecture to 'val', use list to display supported
`-marm`	generate arm code
`-mbe8`	big endian target
`-mbig-endian`	big endian target
`-mcmse`	v8m architecture has cmse instructions
`-mcpu=val`	set cpu core to 'val', use list to display supported
`-memorymap` file	supply memory map file in 'file'
`-memorymapmacros` macros	define macros for memory map file in 'macros'
`-mfloat-abi=val`	specify the floating-point abi to use, val can be 'soft', 'softfp', 'hard'
`-mfp16-format=ieee`	specify the format of the __fp16 half-precision floating-point type
`-mfpu=val`	set fpu to 'val', use list to display supported
`-mlittle-endian`	little endian target
`-mno-thumb-interwork`	do not generate interworking code for v4t architecture
`-mno-unaligned-access`	disable unaligned word and half-word load/store instructions
`-mthumb`	generate thumb code, default is to generate ARM code for processors that support it
`-mtp=soft`	specify the thread local storage model
`-munaligned-access`	enable unaligned word and half-word load/store instructions
`-n`	show commands but don't execute them
`-nodefaultlibs`	disable linking of standard libraries
`-noshortenums`	enumerations are int sized
`-noshortwchar`	wide characters are 32-bit
`-nostderr`	redirect output from stderr to stdout
`-nostdinc`	do not search any standard directories for include files
`-nostdlib`	disable linking of standard libraries
`-nowarn-enumsize`	no linker warning on mismatched enum sized input files
`-nowarn-mismatch`	no linker warning on mismatched architecture input files
`-nowarn-rwx-segments`	no linker warning on load segments with RWX permissions
`-nowarn-wcharsize`	no linker warning on mismatched wchar sized input files

-o file	leave output in 'file'
-O0	set optimization level to level 0
-O1	set optimization level to level 1
-O2	set optimization level to level 2
-O3	set optimization level to level 3
-Os	set optimization level to optimize for size
-Oz	set optimization level to optimize for more size
-patch cmd	run 'cmd' after link but before the creation of the additional output file
-pch	generate a precompiled header file
-pedantic	warning on non-standard language usage
-pedantic-errors	error on non-standard language usage
-placement file	supply placement file in 'file'
-placementmacros macros	define macros for placement file in 'macros'
-placementsegments segments	memory segments for placement in 'segments'
-printf=d[ll][w]	double, optional long long, optional wchar
-printf=f[ll][w]	float, optional long long, optional wchar
-printf=i[p][w]	integer, optional width and precision, optional wchar
-printf=ll[p][w]	long long integer, optional width and precision, optional wchar
-Rc,name	name the default code section to 'name'
-Rd,name	name the default data section to 'name'
-Rk,name	name the default const section to 'name'
-rtti	enable C++ rtti
-Rz,name	name the default bss section to 'name'
-scanf=d[ll][c]	double, optional long long, optional %[...] and %[^...] character class
-scanf=ll[c]	long long integer, optional %[...] and %[^...] character class
-segger	use SEGGER assembler/compiler/lto
-segger-ld	use SEGGER linker
-shortenums	enumerations are minimal container sized
-shortwchar	wide characters are 16-bit
-simd=neon	generate simd vector processing code
-stack-sizes	generate stack-sizes section

`-std=s`	set language standard to 's', use list to display supported
`-stop`	generate stack overflow protection code
`-stripdebug`	strip debug information from linked executable
`-stripsymbols`	strip symbols from linked executable
`-symbols=s`	link symbols file 's' into executable
`-Tfile`	use 'file' as linker script
`-thumb`	generate thumb code, default is to generate ARM code for processors that support it
`-unwindtables`	generate stack unwind tables
`-v`	show command lines as they are executed
`-vectorize`	enable auto vectorization code generation
`-version`	show version
`-W`	supply option to the compiler
`-w`	suppress warnings
`-Wa,x`	pass 'x' to the assembler
`-Wc,x`	pass 'x' to the compiler
`-we`	treat warnings as errors
`-Werror`	treat warnings as errors
`-Wl,x`	pass 'x' to the linker
`-x t`	subsequent files are considered to be of file type 't'
`-xa`	subsequent files are considered to be library files
`-xasm`	subsequent files are considered to be assembly code
`-xassembler-with-cpp`	subsequent files are considered to be assembly code
`-xc`	subsequent files are considered to be C code
`-xc++`	subsequent files are considered to be C++ code
`-Xlinker x`	pass 'x' to the linker
`-xo`	subsequent files are considered to be object code

Command-Line Project Builder

emBuild is a program used to build your software from the command line without using **SEGGER** Embedded Studio. You can, for example, use **emBuild** for nightly (automated) builds, production builds, and batch builds.

Building with a SEGGER Embedded Studio project file

You can specify a SEGGER Embedded Studio project file:

Syntax

emBuild [*options*…] *project-file*

You must specify a configuration to build using **-config.** For instance:

```
emBuild -config "V5T Thumb LE Release" arm.emProject
```

The above example uses the configuration **V5T Thumb LE Release** to build all projects in the solution contained in **arm.emProject**.

To build a specific project that is in a solution, you can specify it using the **-project** option. For example:

```
emBuild -config "V5T Thumb LE Release" -project "libm" libc.emProject
```

This example will use the configuration **V5T Thumb LE Release** to build the project **libm** that is contained in **libc.emProject**.

If your project file imports other project files (using the <import…> mechanism), when denoting projects you must specify the solution names as a comma-separated list in parentheses after the project name:

```
emBuild -config "V5T Thumb LE Release" -project "libc(C Library)" arm.emProject
```

libc(C Library) specifies the **libc** project in the **C Library** solution that has been imported by the project file **arm.emProject**.

To build a specific solution that has been imported from other project files, you can use the **-solution** option. This option takes the solution names as a comma-separated list. For example:

```
emBuild -config "ARM Debug" -solution "ARM Targets,EB55" arm.emProject
```

In this example, **ARM Targets,EB55** specifies the **EB55** solution imported by the **ARM Targets** solution, which was itself imported by the project file **arm.emProject**.

You can do a batch build using the **-batch** option:

```
emBuild -config "ARM Debug" -batch libc.emProject
```

This will build the projects in **libc.emProject** that are marked for batch build in the configuration **ARM Debug**.

By default, a *make-style* build will be done—i.e., the dates of input files are checked against the dates of output files, and the build is avoided if the output is up to date. You can force a complete build by using the **-rebuild** option. Alternatively, to remove all output files, use the **-clean** option.

To see the commands being used in the build, use the **-echo** option. To also see why commands are being executed, use the **-verbose** option. You can see what commands will be executed, without executing them, by using the **-show** option.

Building without a SEGGER Embedded Studio project file

To use **emBuild** without a SEGGER Embedded Studio project, specify the name of an installed project template, the name of the project, and the files to build. For example:

```
emBuild -config … -template LM3S_EXE -project myproject -file main.c
```

Or, instead of a template, you can specify a project type:

```
emBuild -config … -type "Library" -project myproject -file main.c
```

You can specify project properties with the **-property** option:

```
emBuild … -property Target=LM3S811
```

Options:

Option	Description
`-batch`	batch build
`-clean`	remove all output and intermediate files
`-config` 'name'	(batch) build in 'name' configuration
`-D` 'name'='value'	set global macro $('name') to 'value'
`-echo`	show command lines as they are executed
`-export-build`	show the command lines and dependencies
`-file` 'name'	add file 'name' to template-created project
`-help`	show help text
`-keepgoing`	keep building when errors occur
`-nostderr`	redirect output from stderr to stdout
`-nounity`	disable unity building
`-packagesdir` 'name'	set $(PackagesDir) to 'name'
`-project` 'name'	(batch) build 'name' project
`-property` 'name'='value'	set project property 'name' to 'value'
`-rebuild`	always do the build steps
`-sconfig` 'name'	solution properties are set in 'name' configuration
`-show`	show the build steps but don't execute them

`-solution` 'name'	(batch) build 'name' solution
`-sproperty` 'name'='value'	set solution property 'name' to 'value'
`-studiodir` 'name'	set $(StudioDir) to 'name'
`-template` 'name'	create the project from the template 'name'
`-templatesfile` 'path'	get project template files from 'path'
`-threadnum` 'n'	use 'n' threads for build
`-time`	show time taken
`-type` 'type'	create the project from the project 'type'
`-verbose`	show build information

Command-Line Simulator

emSim is a program that allows you to run SEGGER Embedded Studio's instruction set simulator from the command line.

The primary purpose of **emSim** is to enable command line tests to be run. The debug I/O provided by SEGGER Embedded Studio is supported, as are command line arguments and exit.

emSim will accept a single elf file, it will allocate and load memory regions based on the program sections in the elf file. emSim will start execution from the entry point symbol contained in the elf file. emSim will terminate when exit is called or execution reaches a specified symbol.

Example

Assuming that app.c contains the following

```
#include <stdio.h>
#include <stdlib.h>
int
main(int argc, const char *argv[])
{
  int i;
  for (i = 1; i < argc; i++)
    printf("argv[%d]=%s\n", i, argv[i]);
  exit(EXIT_SUCCESS);
}
```

and app.elf has been built with the preprocessor definition FULL_LIBRARY then

```
emSim app.elf hello world
```

will produce the output

```
argv[1]=hello
argv[2]=world
```

if the debug I/O implementation has to set breakpoints or poll memory locations then you can supply the name of the symbol to breakpoint on that will enable the debug I/O

```
emSim app.elf -startup __startup_complete hello world
```

if the application uses memory that isn't allocated in the elf file then you can supply the memory segments that the simulator should create

```
emSim app.elf -segments 0x08000000;0x10000;0x20000000;0x10000
```

and there is an alternative form of this

```
emSim app.elf -memory-segments "FLASH RX 0x08000000 0x10000;SRAM RWX 0x20000000 0x10000"
```

The simulator attempts to determine the machine architecture from data in the elf file this can be override using

If the simulator doesn't support the architecture then the the list of supported architectures will be displayed.

The simulator will start executing at the entry point symbol in the elf file. If the application doesn't set the stack pointer then you can supply this

```
emSim app.elf -stackpointer __stack_end__
```

If the application doesn't call exit then you can supply a symbol to breakpoint on that will terminate the simulation

```
emSim app.elf -end _Exit
```

You can show an instruction trace

```
emSim app.elf -trace
```

Which will show the addresses of instructions, the instruction opcode and the disassembly

```
0x000002d8 E59F003C    ldr r0, =0xE01FC000
0x000002dc E3A01000    mov r1, #0
0x000002e0 E5801000    str r1, [r0]
0x000002e4 E3A01003    mov r1, #3
0x000002e8 E5801004    str r1, [r0, #4]
0x000002ec E3A01002    mov r1, #2
```

You can display the instruction execution counts at the end of the simulation

```
emSim app.elf -counts
```

Which will show the addresses of instructions, with the number of times executed

```
0x08000298=1
0x0800029a=1
...
0x080008ac=278
0x080008ae=2
...
```

You can supply a list of memory word values to simulate ROM or hardware status registers

```
emSim app.elf -values "0xE0082000=0xFEEDFACE"
```

Which will always return 0XFEEDFACE when the address 0xE0082000 is read or fetched.

Similarly to you can supply a list of memory half word values

```
emSim app.elf -values2 "0xE0082000=0xFACE;0xE0082000=0xFEED"
```

The address is evaluated as a debug expression so you can supply symbol names

```
emSim app.elf -values2 "SystemInit=0x8082"
```

Which will cause the simulator to execute a RISC-V return instruction when SystemInit is called.

Usage:

`emSim` *file* [*options*] *args* …

Option	Description
`-arch` 'a'	Specify architecture to simulate
`-arch` list	List supported architectures
`-counts`	Show execution counts when the simulation ends
`-end` 's'	Specify the symbol to end simulation
`-max` 'c'	Specify the maximum number of instructions to simulate
`-memory-segments` 'name [access] start size;…'	Specify the list of memory segments
`-segments` 'start;size;…'	Specify the list of memory segments
`-stackpointer` 's'	Specify the starting stackpoint symbol
`-startup` 's'	Specify the startup completion point symbol
`-trace`	Show instruction execution
`-values` 'address=value;…'	Specify a list of memory word values to simulate
`-values2` 'address=value;…'	Specify a list of memory half word values to simulate

Command-Line Scripting

emScript is a program that allows you to run SEGGER Embedded Studio's JavaScript (ECMAScript) interpreter from the command line.

The primary purpose of **emScript** is to facilitate the creation of platform-independent build scripts.

Syntax

emScript [*options*] *file*…

Command-line options

This section describes the command-line options accepted by emScript.

-define (Define global variable)

Syntax

-define *variable=value*

Description

-help (Show usage)

Syntax

-help

Description

Display usage information and command line options.

-load (Load script file)

Syntax

-load *path*

Description

Loads the script file *path*.

-define (Verbose output)

Syntax

-verbose

Description

Produces verbose output.

emScript classes

emScript provides the following predefined classes:

- **BinaryFile**
- **CWSys**
- **WScript**

Example uses

The following example demonstrates using **emScript** to increment a build number:

First, add a JavaScript file to your project called `incbuild.js` containing the following code:

```
function incbuild()
{
  var file = "buildnum.h"
  var text = "#define BUILDNUMBER "
  var s = CWSys.readStringFromFile(file);
  var n;
  if (s == undefined)
    n = 1;
  else
    n = eval(s.substring(text.length)) + 1;
  CWSys.writeStringToFile(file, text + n);
}

// Executed when script loaded.
incbuild();
```

Add a file called `getbuildnum.h` to your project containing the following code:

```
#ifndef GETBUILDNUM_H
#define GETBUILDNUM_H

unsigned getBuildNumber();

#endif
```

Add a file called `getbuildnum.c` to your project containing the following code:

```
#include "getbuildnum.h"
#include "buildnum.h"

unsigned getBuildNumber()
{
  return BUILDNUMBER;
}
```

Now, to combine these:

- Set the **Build Options** > **Always Rebuild** project property of `getbuildnum.c` to **Yes**.
- Set the **User Build Step Options** > **Pre-Compile Command** project property of `getbuildnum.c` to **"$(StudioDir)/bin/emScript" -load "$(ProjectDir)/incbuild.js"**.

Embed

Embed is a program that converts a binary file into a C/C++ array definition.

The primary purpose of the **Embed** tool is to provide a simple method of embedding files into an application. This may be useful if you want to include firmware images, bitmaps, etc. in your application without having to read them first from an external source.

Syntax

embed *variable_name input_file output_file*

variable_name is the name of the C/C++ array to be initialised with the binary data.

input_file is the path to the binary input file.

output_file is the path to the C/C++ source file to generate.

Example

To convert a binary file *image.bin* to a C/C++ file called *image.h*:

```
embed img image.bin image.h
```

This will generate the following output in *image.h*:

```
static const unsigned char img[] = {
  0x5B, 0x95, 0xA4, 0x56, 0x16, 0x5F, 0x2D, 0x47,
  0xC5, 0x04, 0xD4, 0x8D, 0x73, 0x40, 0x31, 0x66,
  0x3E, 0x81, 0x90, 0x39, 0xA3, 0x8E, 0x22, 0x37,
  0x3C, 0x63, 0xC8, 0x30, 0x90, 0x0C, 0x54, 0xA4,
  0xA2, 0x74, 0xC2, 0x8C, 0x1D, 0x56, 0x57, 0x05,
  0x45, 0xCE, 0x3B, 0x92, 0xAD, 0x0B, 0x2C, 0x39,
  0x92, 0x59, 0xB9, 0x9D, 0x01, 0x30, 0x59, 0x9F,
  0xC5, 0xEA, 0xCE, 0x35, 0xF6, 0x4B, 0x05, 0xBF
};
```

Command-Line License Manager

The **emLicense** program can be used to install, remove and list licenses from the command line.

Usage	Description
`emLicense` install *license*-strings...	Install licenses
`emLicense` list	List all installed licenses
`emLicense` remove *license*-ids...	Remove licenses

Option	Description
`-erase-all-existing`	Erase all existing licenses when installing
`-verbose`	Produce verbose output

The **install** command can be used to install licenses:

```
$ emLicense install license-strings...
```

The **list** command can be used to list installed licenses:

```
$ emLicense list
Product: SEGGER Embedded Studio
Ethernet MAC Address Lock: 01-02-03-04-05-06 (OK)
Licensed to: Joe Bloggs
License ID: 00010203-04050607-08090A0B-0C0D0E0F
```

The **remove** command can be used to uninstall licenses:

```
$ emLicense remove 00010203-04050607-08090A0B-0C0D0E0F
```

Linker script file generator

The command line program **mkld** generates a GNU ld linker script from a SEGGER Embedded Studio memory map or section placement file.

Syntax

- **mkld** *-memory-map-file inputfile outputfile* [*options*…]
- **mkld** *-memory-map-segments segments outputfile* [*options*…]

Description

- **inputfile** is the name of the SEGGER Embedded Studio memory map file to generate the ld script from.
- **segments** is a list of memory segments of the form *SegmentName RWX Address Size*
- **outputfile** is the the name of the ld script file to write.

Command-line options

This section describes the command-line options accepted by *mkld*.

-check-section-overflow

Syntax

-check-section-overflow

Description

Add checks for memory section overflow to the linker script.

-check-segment-overflow

Syntax

-check-segment-overflow

Description

Add checks for memory segment overflow to the linker script.

-disable-missing-runin-error

Syntax

-disable-missing-runin-error

Description

Discard any sections with a missing run in section.

-memory-map-macros

Syntax

-memory-map-macros *macro*=*value*[;*macro*=*value*]

Description

Define SEGGER Embedded Studio macros to use when reading a memory map file.

-no-check-unplaced-sections

Syntax

-no-check-unplaced-sections

Description

Removes checks for unplaced memory sections from the linker script.

-no-ctors

Syntax

-no-ctors

Description

Ignore the .ctors section.

-no-dtors

Syntax

-no-ctors

Description

Ignore the .dtors section.

-section-placement-file

Syntax

-section-placement-file *filename*

Description

Generate a GNU ld linker script from the SEGGER Embedded Studio section placement file *filename*. If this option is used, a memory map file should also be specified with the *-memory-map-file* option.

-section-placement-macros

Syntax

-section-placement-macros *macro*=*value*[*;macro*=*value*]

Description

Define SEGGER Embedded Studio macros to use when reading a section placement file.

-symbols

Syntax

-symbols *symbol*=*value*[;*symbol*=*value*]

Description

Add extra symbol definitions to the ld linker script.

Package Generator

To create a package the program **mkpkg** can be used. The set of files to put into the package should be in the desired location in the `$(PackagesDir)` directory. The **mkpkg** command should be run with `$(PackagesDir)` as the working directory and all files to go into the package must be referred to using relative paths. A package must have a package description file that is placed in the `$(PackagesDir)/packages` directory. The package description file name must end with `_package.xml`. If a package is to create entries in the new project wizard then it must have a file name `project_templates.xml`.

For example, a package for the mythical FX150 processor would supply the following files:

- A project template file called `targets/FX150/project_templates.xml`. The format of the project templates file is described in **Project Templates file format**.
- The `$(PackagesDir)`-relative files that define the functionality of the package.
- A package description file called `packages/FX150_package.xml`. The format of the package description file is described in **Package Description file format**.

The package file `FX150.hzq` would be created using the following command line:

```
mkpkg -c packages/FX150.hzq targets/FX150/project_templates.xml … packages/
FX150_package.xml
```

You can exclude specific files or directories from being added to a package using the **-exclude** option:

```
mkpkg -c packages/FX150.hzq targets/FX150 -exclude targets/FX150/excluded\file.txt -exclude
 targets/FX150/excluded_directory packages/FX150_package.xml
```

You can list the contents of the package using the **-t** option:

```
mkpkg -t packages/FX150.hzq
```

You can remove an entry from a package using the **-d** option:

```
mkpkg -d packages/FX150.hzq -d filename
```

You can add or replace a file into an existing package using the **-r** option:

```
mkpkg -r packages/FX150.hzq -r filename
```

You can extract files from an existing package using the **-x** option:

```
mkpkg -x packages/FX150.hzq output_directory
```

You can automate the package creation process using a **Combining** project type.

- Using the new project wizard create a combining project in the directory `$(PackagesDir)`
- Set the **Output File Path** property to be `$(PackagesDir)/packages/mypackage.hzq`
- Set the **Combine command** property to `$(StudioDir)/bin/mkpkg -c $(CombiningOutputFilePath) $(CombiningRelInputPaths)`

- Add the files you want to go into the package into the project using the Project Explorer
- Right-click the project node in the Project Explorer and choose **Build**

When a package is installed, the files in the package are copied into the desired `$(PackagesDir)`-relative locations. When a file is copied into the `$(PackagesDir)/packages` directory and its filename ends with `_package.xml` the file `$(PackagesDir)/packages/installed_packages.xml` is updated with an entry:

```
<include filename="FX150_package.xml" />
```

During development of a package you can manually edit this file. The same applies to the file `$(PackagesDir)/targets/project_templates.xml` which will contain a reference to your `project_templates.xml` file.

Usage:

`mkpkg` *[options] package_filename file1 file2 …`*

Option	Description
`-c`	Create a new package.
`-compress` *level*	Change compression level (0 for none, 9 for maximum).
`-d`	Remove files from a package.
`-exclude` *path*	Exclude path when adding files to a package.
`-f`	Output files to stdout.
`-overwrite`	Overwrite existing files.
`-no-date`	Do not add date attribute to package.
`-r`	Replace files in a package.
`-readonly`	Force all files to have read only attribute.
`-set-attr` *attribute=value*	Set package attribute to value.
`-sub-arch-endian`	Create architecture and endian specific sub packages.
`-sub-arch-endian-compatiblity`	Create architecture and endian specific sub packages including compatibility packages for versions of the IDE that don't have `$(LibEndian)` macro.
`-sub-base-type` *type*	Specify the type description of the base package.
`-sub-no-description-modify`	Do not modify sub package descriptions.
`-sub-regexp` *name suffix regexp*	Create sub package containing files matching regular expression.
`-sub-type` *type*	Specify the type description of the sub packages.
`-t`	List the contents of a package.
`-v`	Be chatty.

Option	Description
-V	Show version information.
-x	Extract files from a package.

See also **Command-line Package Manager**.

Command-Line Package Manager

The **pkg** program can be used to download, install, remove and search for packages from the command line.

Usage	Description
`pkg history` *package_names*...	List version history of packages
`pkg install` *package_names*...	Download and install packages
`pkg install -manual` *package_files*...	Manually install package files
`pkg list`	List all available packages
`pkg list` -installed	List installed packages
`pkg list` -installed-names	List installed package names
`pkg list` -dependencies *package_names*...	List package dependencies
`pkg list` -dependents *package_names*...	List dependent packages
`pkg remove` *package_names*...	Remove packages
`pkg remove` -all	Remove all packages
`pkg search` *keywords*...	Search for packages
`pkg update`	Update list of available packages
`pkg upgrade`	Upgrade all installed packages
`pkg upgrade` *package_names*...	Upgrade selected packages

Option	Description
`-D` *macro*=value	Set a global macro
`-keepgoing`	Continue when errors occur
`-legacy`	Include legacy packages
`-nodelete`	Don't delete downloaded packages after installation
`-noverify`	Don't verify downloaded packages
`-outputformat` *string*	Specify list/search output format string
`-packagesdir` *directory*	Set the packages directory to be *directory*
`-packagesurl` *url*	Set the URL of the packages website to be *url*
`-quiet`	Do not output any progress messages
`-rootuserdir` *directory*	Set the root user data directory to *directory*
`-verbose`	Produce verbose output
`-yes`	Answer yes to all questions without prompting

Macro	Description
`$(Description)`	Package description
`$(Name)`	Package name

Macro	Description
`$(Title)`	Package title
`$(Version)`	Package version

Before you can download, install or search for packages you must first update the local list of available packages:

```
$ pkg update
```

The **search** command can be used to search for a specific package:

```
$ pkg search libcxx
libcxx_arm - ARM libcxx Library Package (1.1)
```

The **install** command can be used to install a package:

```
$ pkg install libcxx_arm
```

The **list** command can be used to list installed packages:

```
$ pkg list -installed
libcxx_arm - ARM libcxx Library Package (1.1)
```

The **history** command can be used to show package history:

```
$ pkg history libcxx_arm
libcxx_arm - libcxx Library Package [ARM]

  1.1 (Installed)
    - Fixed name of Type Interpretation File.

  1.0
    - Initial release.
```

Specific versions of a package can be installed:

```
$ pkg install libcxx_arm:1.0
```

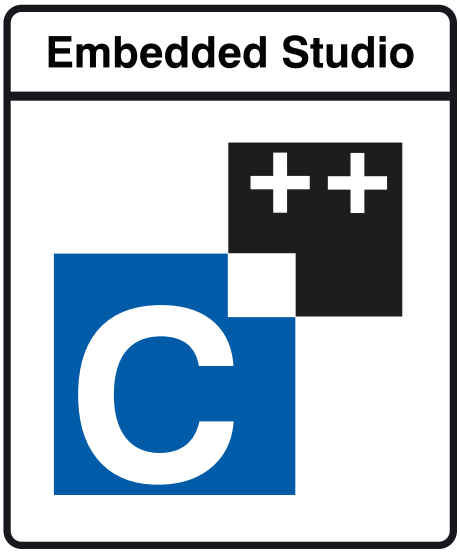
The **upgrade** command can be used to upgrade to the latest version of a package:

```
$ pkg upgrade libcxx_arm
```

The **remove** command can be used to uninstall a package:

```
$ pkg remove libcxx_arm
```

See also **Package Generator**.



Appendices

File formats

This section describes the file formats SEGGER Embedded Studio uses:

Memory Map file format

Describes the memory map file format that defines memory regions and registers in a microcontroller.

Section Placement file format

Describes the section placement file format that maps program sections to memory areas in the target microcontroller.

Project file format

Describes the format of SEGGER Embedded Studio project files.

Project Templates file format

Describes the format of project template files used by the **New Project** wizard.

Property Groups file format

Describes the format of the property groups file you can use to define 'meta-properties'.

Package Description file format

Describes the format of the package description files you use to create packages other users can install in SEGGER Embedded Studio.

External Tools file format

Describes the format of external tool configuration files you use to extend SEGGER Embedded Studio.

Debugger Type Interpretation file format

Describes the format of the debugger type interpretation file.

Memory Map file format

SEGGER Embedded Studio memory-map files are structured using XML syntax for its simple construction and parsing.

The first entry of the project file defines the XML document type used to validate the file format.

```
<!DOCTYPE Board_Memory_Definition_File>
```

The next entry is the `Root` element. There can only be one `Root` element in a memory map file:

```
<Root name="My Board">
```

A `Root` element has a `name` attribute — every element in a memory map file has a `name` attribute. Names should be unique within a hierarchy level. Within a `Root` element, there are `MemorySegment` elements that represent regions within the memory map.

```
<Root name="My Board">
  <MemorySegment name="Flash" start="0x1000" size="0x200" access="ReadOnly">
```

`MemorySegment` elements have the following attributes:

- *start*: The start address of the memory segment. A simple expression, usually a hexadecimal number with a 0x prefix.
- *size*: The size of the memory segment. A simple expression, usually a hexadecimal number with a 0x prefix.
- *access*: The permissible access types of the memory segment. One of `ReadOnly`, `Read/Write`, `WriteOnly`, or `None`.
- *address_symbol*: A symbolic name for the start address of the memory segment.
- *size_symbol*: A symbolic name for the size of the memory segment.
- *end_symbol*: A symbolic name for the end address of the memory segment.

`RegisterGroup` elements are used to organize registers into groups. `Register` elements are used to define peripheral registers:

```
<Root name="My Board" >
 <MemorySegment name="System" start="0x2000" size="0x200" >
   <RegisterGroup name="Peripheral1" start="0x2100" size="0x10" >
     <Register name="Register1" start="+0x8" size="4" >
```

`RegisterGroup` elements have the same attributes as `MemorySegment` elements. `Register` elements have the following attributes:

- *name*: Register names should be valid C/C++ identifier names, i.e., alphanumeric characters and underscores are allowed but names cannot start with a number.
- *start*: The start address of the memory segment. Either a C-style hexadecimal number or, if given a + prefix, an offset from the enclosing element's start address.

- *size*: The size of the register in bytes, either 1, 2, or 4.
- *access*: The same as the `access` attribute of the `MemorySegment` element.
- *address_symbol*: The same as the `address_symbol` attribute of the `MemorySegment` element.

A `Register` element can contain `BitField` elements that represent the bits in a peripheral register:

```
<Root name="My Board" >
 <MemorySegment name="System" start="0x2000" size="0x200" >
   <RegisterGroup name="Peripheral1" start="0x2100" size="0x10" >
     <Register name="Register1" start="+0x8" size="4" >
       <BitField name="Bits_0_to_3" start="0" size="4" />
```

`BitField` elements have the following attributes:

- *name*: The same as the `name` attribute of the `RegisterGroup` element.
- *start*: The starting bit position, 0–31.
- *size*: The total number of bits, 1–32.

A `Bitfield` element can contain `Enum` elements:

```
<Root name="My Board" >
  <RegisterGroup name="Peripheral1" start="0x2100" size="0x10" >
    <Register name="Register1" start="+0x8" size="4" >
      <BitField name="Bits_0_to_3" start="0" size="4" />
        <Enum name="Enum3" start="3" />
        <Enum name="Enum5" start="5" />
```

You can import CMSIS SVD files (see **http://www.onarm.com/**) into a memory map using the `ImportSVD` element:

```
<ImportSVD filename="$(TargetsDir)/targets/Manufacturer1/Processor1.svd.xml">
```

The `filename` attribute is an absolute filename which is macro-expanded using SEGGER Embedded Studio system macros.

When a memory map file is loaded either for the memory map viewer or to be used for linking or debugging, it is preprocessed using the (as yet undocumented) SEGGER Embedded Studio XML preprocessor.

Section Placement file format

SEGGER Embedded Studio section-placement files are structured using XML syntax to enable simple construction and parsing.

The first entry of the project file defines the XML document type used to validate the file format:

```
<!DOCTYPE Linker_Placement_File>
```

The next entry is the `Root` element. There can only be one `Root` element in a memory map file:

```
<Root name="Flash Placement">
```

A `Root` element has a `name` attribute. Every element in a section-placement file has a `name` attribute. Each name should be unique within its hierarchy level. Within a `Root` element, there are `MemorySegment` elements. These correspond to memory regions defined in a memory map file that will be used in conjunction with the section-placement file when linking a program. For example:

```
 <Root name="Flash Placement">
  <MemorySegment name="FLASH">
```

A `MemorySegment` contains `ProgramSection` elements that represent program sections created by the C/C++ compiler and assembler. The order of `ProgramSection` elements within a `MemorySegment` element represents the order in which the sections will be placed when linking a program. The first `ProgramSection` will be placed first and the last one will be placed last.

```
<Root name="My Board" >
  <MemorySegment name="FLASH">
    <ProgramSection name=".text">
```

`ProgramSection` elements have the following attributes:

- *address_symbol*: A symbolic name for the start address of the section.
- *alignment*: The required alignment of the program section; a decimal number specifying the byte alignment.
- *end_symbol*: A symbolic name for the end address of the section.
- *load*: If **Yes**, the section is loaded. If **No**, the section isn't loaded.
- *runin*: This specifies the name of the section to copy this section to. Multiple sections can be specified separated by a semicolon, the first section that exists will be used.
- *size*: The optional size of the program section in bytes, a hexadecimal number with a 0x prefix. The macro *$(SEGMENT_SIZE_REMAINING)* can be used for size calcuations based on the remaining number of bytes in the segment.
- *size_alignment*: The optional size aligment of the program section in bytes.
- *size_symbol*: A symbolic name for the size of the section.
- *start*: The optional start address of the program section, a hexadecimal number with a 0x prefix.

When a section placement file is used for linking it is preprocessed using the (as yet undocumented) SEGGER Embedded Studio XML preprocessor.

Project file format

SEGGER Embedded Studio project files are held in text files with the `.emProject` extension. Because you may want to edit project files, and perhaps generate them, they are structured using XML syntax to enable simple construction and parsing.

The first entry of the project file defines the XML document type used to validate the file format:

```
<!DOCTYPE CrossStudio_Project_File>
```

The next entry is the `solution` element; there can only be one `solution` element in a project file. This specifies the solution name displayed in the **Project Explorer** and has a version attribute that defines the file-format version of the project file. Solutions can contain projects, projects can contain folders and files, and folders can contain folders and files. This hierarchy is reflected in the XML nesting—for example:

```
<solution version="1" Name="solutionname">
  <project Name="projectname">
    <file Name="filename" />
    <folder Name="foldername">
      <file Name="filename2" />
    </folder>
  </project>
</solution>
```

Note that each entry has a `Name` attribute. Names of `project` elements must be unique to the solution, and names of `folder` elements must be unique to the project, but names of files do not need to unique.

Each `file` element must have a `file_name` attribute that is unique to the project. Ideally, the `file_name` is a file path relative to the project (or solution directory), but you can also specify a full file path, if you want to. File paths are case-sensitive and use "/" as the directory separator. They may contain macro instantiations, so file paths cannot contain the "$" character. For example…

```
<file file_name="$(StudioDir)/source/crt0.s" Name="crt0.s" />
```

…will be expanded using the value of **$(StudioDir)** when the file is referenced from SEGGER Embedded Studio.

Project properties are held in configuration elements with the `Name` attribute of the configuration element corresponding to the configuration name, e.g., "Debug". At a given project level (i.e., solution, project, folder), there can only be one named configuration element—i.e., all properties defined for a configuration are in single configuration element.

```
<project Name="projectname">
  #
  <configuration project_type="Library" Name="Common" />
  <configuration Name="Release" build_debug_information="No" />
  #
</project>
```

You can use the `import` element to link projects:

```
<import file_name="target/libc.emProject" />
```

Project Templates file format

The SEGGER Embedded Studio **New Project** dialog works from a file called `project_templates.xml` in the `targets` subdirectory of the SEGGER Embedded Studio installation directory. Because you may want to add your own new project types, they are structured using XML syntax to enable simple construction and parsing.

The first entry of the project file defines the XML document type used to validate the file format:

```
<!DOCTYPE Project_Templates_File>
```

The next entry is the `projects` element, which is used to group a set of new project entries into an XML hierarchy.

```
<projects>
  <project> …
</projects>
```

Each entry has a `project` element that contains the class of the project (attribute `caption`), the name of the project (attribute `name`), its type (attribute `type`) and a description (attribute `description`). For example:

```
<project caption="ARM Evaluator7T" name="Executable"
  description="An executable for an ARM Evaluator7T." type="Executable"/>
```

The project type can be one of these:

- *Executable*: — a fully linked executable.
- *Library*: — a static library.
- *Object file*: — an object file.
- *Staging*: — a staging project.
- *Combining*: — a combining project.
- *Externally Built Executable*: — an externally built executable.

The configurations to be created for the project are defined using the `configuration` element, which must have a `name` attribute:

```
<configuration name="ARM RAM Release"/>
```

The property values to be created for the project are defined using the `property` element. If you have a defined value, you can specify this using the `value` attribute and, optionally, set the property in a defined `configuration`, such as:

```
<property name="target_reset_script" configuration="RAM"
  value="Evaluator7T_ResetWithRamAtZero()" />
```

Alternatively, you can include a property that will be shown to the user, prompting them to supply a value as part of the new-project process.

```
<property name="linker_output_format"/>
```

The folders to be created are defined using the `folder` element. The `folder` element must have a `name` attribute and can also have a `filter` attribute. For example:

```
<folder name="Source Files" filter="c;cpp;cxx;cc;h;s;asm;inc" />
```

The files to be in the project are specified using the `file` element. You can use build-system macros (see **Project macros**) to specify files located in the SEGGER Embedded Studio installation directory. Files will be copied to the project directory or just left as references, depending on the value of the `source` attribute:

```
<file name="main.c" source="$(StudioDir)/samples/Shared/main.c"/>
<file name="$(StudioDir)/source/thumb_crt0.s"/>
```

You can define the set of configurations that can be referred to in the top-level `configurations` element:

```
<configurations>
  <configuration> …
</configurations>
```

This contains the set of all configurations that can be created when a project is created. Each configuration is defined using a `configuration` element, which can define the property values for that configuration. For example:

```
<configuration name="Debug">
  <property name="build_debug_information" value="Yes">
```

Property Groups file format

The SEGGER Embedded Studio project system provides a means to create new properties that change a number of project property settings and can also set C pre-processor definitions when selected. Such properties are called *property groups* and are defined in a property-groups file. The property-group file to use for a project is defined by the **Property Groups File** property. These files usually define target-specific properties and are structured using XML syntax to enable simple construction and parsing.

The first entry of the property groups file defines the XML document type, which is used to validate the file format:

```
<!DOCTYPE CrossStudio_Group_Values>
```

The next entry is the `propertyGroups` element, which is used to group a set of property groups entries into an XML hierarchy:

```
<propertyGroups>
  <grouphdots
  #
  <grouphdots
</propertyGroups>
```

Each group has the name of the group (attribute `name`), the name of the options category (attribute `group`), short (attribute `short`) and long (attribute `long`) help descriptions, and a default value (attribute `default`). For example:

```
<group short="Target Processor" group="Build Options" short="Target Processor"
  long="Select a set of target options" name="Target" default="STR912FW44" />
```

Each group has a number of `groupEntry` elements that define the enumerations of the group.

```
<group…\>
  <groupEntry>…
  …
  <groupEntry>…
</group>
```

Each `groupEntry` has the name of the entry (attribute `name`), e.g.:

```
<groupEntry name="STR910FW32">
```

A `groupEntry` has the property values and C pre-processor definitions that are set when the `groupEntry` is selected; they are specified with `property` and `cdefine` elements. For example:

```
<groupEntry>…
  <property>…
  <cdefine>…
  <property>…
</groupEntry>
```

A `property` element has the property's name (attribute `name`), its value (attribute `value`), and an optional configuration (attribute `configuration`):

```
<property name="linker_memory_map_file"
  value="$(StudioDir)/targets/ST_STR91x/ST_STR910FM32_MemoryMap.xml" />
```

A `cdefine` element has the C preprocessor name (attribute `name`) and its value (attribute `value`):

```
<cdefine value="STR910FM32" name="TARGET_PROCESSOR" />
```

Package Description file format

Package-description files are XML files used by SEGGER Embedded Studio to describe a support package, its contents, and any dependencies it has on other packages.

Each package file must contain one `package` element that describes the package. Optionally, the `package` element can contain a collection of `file`, `history`, and `documentation` elements to be used by SEGGER Embedded Studio for documentation purposes.

The filename of the package-description file should match that of the package and end in "*_package.xml*".

Below is an example of two package-description files. The first is for a base chip-support package for the LPC2000; the second is for a board-support package dependent on the first:

Philips_LPC2000_package.xml

```
<!DOCTYPE CrossStudio_Package_Description_File>
<package cpu_manufacturer="Philips" cpu_family="LPC2000" version="1.1" ses_versions="8:1-"
 author="SEGGER" >
  <file file_name="$(TargetsDir)/Philips_LPC210X/arm_target_Philips_LPC210X.htm"
 title="LPC2000 Support Package Documentation" />
  <file file_name="$(TargetsDir)/Philips_LPC210X/Loader.emProject" title="LPC2000 Loader
 Application Solution" />
  <group title="System Files">
    <file file_name="$(TargetsDir)/Philips_LPC210X/Philips_LPC210X_Startup.s" title="LPC2000
 Startup Code" />
    <file file_name="$(TargetsDir)/Philips_LPC210X/Philips_LPC210X_Target.js" title="LPC2000
 Target Script" />
  </group>
  <history>
    <version name="1.1" >
      <description>Corrected LPC21xx header files and memory maps to include GPIO ports 2
 and 3.</description>
      <description>Modified loader memory map so that .libmem sections will be placed
 correctly.</description>
    </version>
    <version name="1.0" >
      <description>Initial Release.</description>
    </version>
  </history>
  <documentation>
    <section name="Supported Targets">
      <p>This CPU support package supports the following LPC2000 targets:
        <ul>
          <li>LPC2103</li>
          <li>LPC2104</li>
          <li>LPC2105</li>
          <li>LPC2106</li>
          <li>LPC2131</li>
          <li>LPC2132</li>
          <li>LPC2134</li>
          <li>LPC2136</li>
          <li>LPC2138</li>
        </ul>
      </p>
    </section>
  </documentation>
```

```
</package>
```

CrossFire_LPC2138_package.xml

```
<!DOCTYPE CrossStudio_Package_Description_File>
<package cpu_manufacturer="Philips" cpu_family="LPC2000" cpu_name="LPC2138"
 board_manufacturer="Rowley Associates" board_name="CrossFire LPC2138"
 dependencies="Philips_LPC2000" version="1.0">
  <file file_name="$(SamplesDir)/CrossFire_LPC2138/CrossFire_LPC2138.emProject"
 title="CrossFire LPC2138 Samples Solution" />
  <file file_name="$(SamplesDir)/CrossFire_LPC2138/ctl/ctl.emProject" title="CrossFire
 LPC2138 CTL Samples Solution" />
</package>
```

Package elements

The `package` element describes the support package, its contents, and any dependencies it has on other packages. Valid attributes for this element are:

Attribute	Description
`author`	The author of the package.
`board_manufacturer`	The manufacturer of the board supported by the package *(if omitted, CPU manufacturer will be used)*.
`board_name`	The name of the specific board supported by the package *(only required for board-support packages)*.
`company_name`	The name of the company to group the package under in the package dialogs. *(if omitted, the Board/CPU manufacturer will be used)*.
`cpu_family`	The family name of the CPU supported by the package *(optional)*.
`cpu_manufacturer`	The manufacturer of the CPU supported by the package.
`cpu_name`	The name of the specific CPU supported by the package *(may be omitted if the CPU family is specified)*.
`description`	A description of the package *(optional)*.
`dependencies`	A semicolon-separated list of packages the package requires to be installed in order to work *(optional)*.
`installation_directory`	The directory in which the package should be installed *(optional - if undefined, defaults to "$(PackagesDir)")*.
`replaces`	A semicolon-separated list of package names listing the packages that this package replaces. The replaced packages are marked as legacy packages and are only visible in the package manager if the if the *Include Legacy Packages* option is selected *(optional)*.

`deprecated`	If set to true, indicates that the package has been deprecated. Deprecated packages are marked as legacy packages and are only visible in the package manager if the if the *Include Legacy Packages* option is selected *(optional)*.
`ses_versions`	A string describing which version of SEGGER Embedded Studio supports the package. The format of the string is *target_id_number:version_range_string*.
`title`	A short description of the package *(optional)*.
`uninstalls`	A semicolon-separated list of packages names listing the packages to be uninstalled if this package is installed *(optional)*.
`version`	The package version number.

File elements

The `file` element is used by SEGGER Embedded Studio for documentation purposes by adding links to files of interest within the package such as example project files and documentation.

Attribute	Description
`file_name`	The file path of the file.
`title`	A description of the file.

Optionally, `file` elements can be grouped into categories using the `group` element.

Group elements

The `group` element is used for categorizing files described by `file` elements into a particular group.

Attribute	Description
`title`	Title of the group.

History elements

The `history` element is used to hold a description of the package's version history.

The `history` element should contain a collection of `version` elements.

Version element

The `version` element is used to hold the description of a particular version of the package.

Attribute	Description
name	The name of the version being described.

The `version` element should contain a collection of `description` elements.

Description elements

Each `description` element contains text that describes a feature of the package version.

Documentation elements

The `documentation` element is used to provide arbitrary documentation for the package.

The `documentation` element should contain a collection of one or more `section` elements.

Section elements

The `section` element contains package documentation in XHTML format.

Attribute	Description
name	The title of the documentation section.

target_id_number

The following table lists the possible target ID numbers:

Target	ID
ARM	8
RISC-V	20

version_range_string

The `version_range_string` can be any of the following:

- *version_number*: The package will only work on *version_number*.
- *version_number-*: The package will work on *version_number* or any future version.
- *-version_number*: The package will work on *version_number* or any earlier version.
- *low_version_number-high_version_number*: The package will work on *low_version_number*, *high_version_number* or any version in between.

External Tools file format

SEGGER Embedded Studio external-tool configuration files are structured using XML syntax for its simple construction and parsing.

Tool configuration files

The SEGGER Embedded Studio application will read the tool configuration file when it starts up. By default, SEGGER Embedded Studio will read the file **$(StudioUserDir)/tools.xml**.

Structure

All tools are wrapped in a **tools** element:

```
<tools>
  #
</tools>
```

Inside the tools element are **item** elements that define each tool:

```
<tools>
  <item name="logical name">
    #
  </item>
</tools>
```

The **item** element requires an **name** attribute, which is an internal name for the tool, and has an optional *wait* element. When SEGGER Embedded Studio invokes the tool on a file or project, it uses the *wait* element to determine whether it should wait for the external tool to complete before continuing. If the *wait* attribute is not provided or is set to *yes*, SEGGER Embedded Studio will wait for external tool to complete.

The way that the tool is presented in SEGGER Embedded Studio is configured by elements inside the

- element.

menu

The **menu** element defines the wording used inside menus. You can place a shortcut to the menu using an ampersand, which must be escaped using **&** in XML, before the shortcut letter. For instance:

```
<menu>&amp;PC-lint (Unit Check)</menu>
```

text

The optional **text** element defines the wording used in contexts other than menus, for instance when the tool appears as a tool button with a label. If **text** is not provided, the tool's textual appearance outside the menu is taken from the **menu** element (and is presented without an shortcut underline). For instance:

```
<text>PC-lint (Unit Check)</text>
```

tip

The optional **tip** element defines the status tip, shown on the status line, when moving over the tool inside SEGGER Embedded Studio:

```
<tip>Run a PC-lint unit checkout on the selected file or folder</tip>
```

key

The optional **key** element defines the accelerator key, or key chord, to use to invoke the tool using the keyboard. You can construct the key sequence using modifiers **Ctrl**, **Shift**, and **Alt**, and can specify more than one key in a sequence (note: Windows and Linux only; OS X does not provide key chords). For instance:

```
<key>Ctrl+L, Ctrl+I</key>
```

message

The optional **message** element defines the text shown in the tool log in SEGGER Embedded Studio when running the tool. For example:

```
<message>Linting</message>
```

match

The optional **match** element defines which documents the tool will operator on. The match is performed using the file extension of the document. If the file extension of the document matches one of the wildcards provided, the tool will run on that document. If there is no **match** element, the tool will run on all documents. For instance:

```
<match>*.c;*.cpp</match>
```

output

The optional **output** element defines the name of the output file created by the tool. If this element is specified the the output file will be opened in the editor when the tool has finished execution. The macros **$(InputPath)** and **$(InputBaseName)** can be used to name the output file. For Instance:

```
<output>$(InputBaseName).txt</output>
```

commands

The **commands** element defines the command line to run to invoke the tool. The command line is expanded using macros applicable to the file derived from the current build configuration and the project settings. Most importantly, the standard **$(InputPath)** macro expands to a full pathname for the target file.

Additional macros constructed by SEGGER Embedded Studio are:

- **$(DEFINES)** is the set of **-D** options applicable to the current file, derived from the current configuration and project settings.
- **$(INCLUDES)** is the set of **-I** options applicable to the current file, derived from the current configuration and project settings.

For instance:

```
<commands>
  &quot;$(LINTDIR)/lint-nt&quot; -i$(LINTDIR)/lnt &quot;$(LINTDIR)/lnt/co-gcc.lnt&quot;
  $(DEFINES) $(INCLUDES) -D__GNUC__ -u -b +macros -w2 -e537 +fie +ffn -width(0,4) -hF1
  &quot;-format=%f:%l:%C:s%t:s%m&quot; &quot;$(InputPath)&quot;
</commands>
```

In this example we intend **$(LINTDIR)** to point to the directly where PC-lint is installed and for **$(LINTDIR)** to be defined as a SEGGER Embedded Studio global macro. You can set global macros using **Tools** > **Options** > **Building** > **Global Macros**.

Note that additional **&quot** entities are placed around pathnames in the **commands** section—this is to ensure that paths that contain spaces are correctly interpreted when the command is executed by SEGGER Embedded Studio.

Debugger Type Interpretation file format

SEGGER Embedded Studio debugger type interpretation files are used by the debugger to provide list and string displays of C++ template container types. The files are structured using XML syntax for its simple construction and parsing.

Consider the following C++ template type

```
template <class _Type> class VeryBasicArray
{
private:
    size_t m_Count;
    _Type *m_pData;
public:
    VeryBasicArray(size_t count)
        : m_Count(count)
       , m_pData(new _Type[count])
    {
    }
}

VeryBasicArray<int> basicArray(5);
```

To display a variable of this type as a list the type interpretation file contains the following entry

```
<List Name="VeryBasicArray&lt;*&gt;"
        Head="(($(T)*)HEAD).m_pData"
        Data="(*($(T0)*)CURRENT)"
        Length="(($(T)*)HEAD).m_Count"
        Next="CURRENT+sizeof($(T0))"/>
```

The **Name** attribute is used to match the template type name note that the **&lt** and **&gt** xml entities are used to match the template argument.

When an entry has been matched the head of the list is located by evaluating the debugger expression in the **Head** attribute. The debugger expressions can contain macros that refer to the matched template type and will use the symbols **HEAD** and **CURRENT**.

The macro **$(T)** refers to the instantiated template type, for the above example **$(T)=VeryBasicArray<int>**.

The template arguments are referred to using macros **$(T0)**, for the above example **$(T0)=int**.

The symbol **HEAD** is the address of the variable being displayed, for the above example if the variable **basicArray** is allocated at address **0x20004000** then the **Head** expression

```
  ((VeryBasicArray<int>*)0x20004000).m_pData
```

will be evaluated by the debugger, note that the . operator and the -> operator are equivalent in debugger expressions.

To display an element the debugger will evaluate the **Data** expression. This expression contains the symbol **CURRENT** which is the address of the element to display, for the above example the first element is at the address **basicArray.m_pData** which is allocated at address **0x20008000** then the **Data** expression

```
(*(int*)0x20008000)
```

will be evaluated by the debugger.

To increment the **CURRENT** symbol the **Next** expression

```
0x20008000+sizeof(int)
```

will be evaluated by the debugger.

Before the **CURRENT** symbol is incremented the debugger needs to check if it is at the end of list. The can be done either as a **Condition** expression or as a **Length** expression

```
((VeryBasicArray<int>*)0x20004000).m_Count
```

The **String** display is simpler than the **List** display since the characters are contiguous and optionally null terminated. The **Data** and **Length** expressions are supported, for example

```
<String Name="string"
        Data="*(($(T) *)HEAD)._M_start_of_storage._M_data"
        Length="(($(T) *)HEAD)._M_finish-(($(T) *)HEAD)._M_start_of_storage._M_data"/>
```

is used to display STLPort std::string types.

Building Environment Options

Build

Property	Description
Automatically Build Before Debug `Environment/Build/Build Before Debug` – Boolean	Enables auto-building of a project before downloading if it is out of date.
Confirm Automatically Build Before Debug `Environment/Build/Show Build Before Debug` – Boolean	Enables the display of the auto-building popup.
Confirm Debugger Stop `Environment/Build/Confirm Debugger Stop` – Boolean	Present a warning when you start to build that requires the debugger to stop.
Display ETA `Environment/Build/Display ETA` – Boolean	Selects whether to attempt to compute and display the ETA on building.
Display Progress Bar `Environment/Build/Display Progress Bar` – Boolean	Selects whether to display progress bar on building.
Echo Build Command Lines `Environment/Build/Show Command Lines` – Boolean	Selects whether build command lines are written to the build log.
Echo Raw Error/Warning Output `Environment/Build/Show Unparsed Error Output` – Boolean	Selects whether the unprocessed error and warning output from tools is displayed in the build log.
Find Error After Building `Environment/Build/Find Error After Build` – Boolean	Moves the cursor to the first diagnostic after a build completes with errors.
Global Macros `Environment/Macros/Global Macros` – StringList	Build macros that are shared across all solutions and projects e.g. paths to library files.
Keep Going On Error `Environment/Build/Keep Going On Error` – Boolean	Build doesn't stop on error.
Save Project File Before Building `Environment/Build/Save Project File On Build` – Boolean	Selects whether to save the project file prior to build.
Show Build Information `Environment/Build/Show Build Information` – Boolean	Show build information.

Build Acceleration

Property	Description
Disable Unity Build `Environment/Build/Disable Unity Build` – Boolean	Ignore Unity Build project properties and always build individual project components.
Parallel Building Threads `Environment/Build/Building Threads` – IntegerRange	The number of threads to launch when building.

Build Log

Property	Description
Highlight Links `Environment/Build Log/Highlight Links` – Boolean	Highlight and navigate URL links in output.
Order `Environment/Build Log/Order` – Enumeration	Specifies whether the build log output is ordered by time or by building thread.
Show Command Lines `Environment/Build Log/Show Command Lines` – Boolean	Selects whether to show or hide lines containing build commands in the build log.
Show Errors `Environment/Build Log/Show Errors` – Boolean	Selects whether to show or hide lines containing errors in the build log.
Show Errors In Vertical Scroll Bar `Environment/Build Log/Show Errors In Vertical Scroll Bar` – Boolean	Annotate the vertical scroll bar with error positions.
Show Notes `Environment/Build Log/Show Notes` – Boolean	Selects whether to show or hide lines containing notes in the build log.
Show Other Lines `Environment/Build Log/Show Other Lines` – Boolean	Selects whether to show or hide unclassified lines in the build log.
Show Warnings `Environment/Build Log/Show Warnings` – Boolean	Selects whether to show or hide lines containing warnings in the build log.
Show Warnings In Vertical Scroll Bar `Environment/Build Log/Show Warnings In Vertical Scroll Bar` – Boolean	Annotate the vertical scroll bar with warning positions.

Window

Property	Description

Show Build Log On Build `Environment/Show Transcript On Build` – Boolean	Show the build log when a build starts.

Debugging Environment Options

Breakpoint

Property	Description
Disassembly Breakpoints `Environment/Debugger/Disassembly Breakpoints` – Boolean	What to do with disassembly breakpoints on debug stop/start.
Focus On Breakpoint `Environment/Debugger/Focus On Breakpoint` – Boolean	Focus IDE when breakpoint is hit.

Debug Terminal

Property	Description
Debug Terminal Command Line `Debug Terminal/Debug Terminal Command Line` – String	Specifies the command line to use when debugging a program that runs in an external terminal.
Debug Terminal TTY `Debug Terminal/TTY` – String	Specifies the path to the terminal device to use as debug terminal. For example, '/dev/pts/1'.

Display

Property	Description
Close Disassembly On Mode Switch `Environment/Debugger/Close Disassembly On Mode Switch` – Boolean	Close Disassembly On Mode Switch.
Data Tips Display a Maximum Of `Environment/Debugger/Maximum Array Elements Displayed` – IntegerRange	Selects the maximum number of array elements displayed in a data tip.
Default Display Mode `Environment/Debugger/Default Variable Display Mode` – Enumeration	Selects the format that data values are shown in.
Display Floating Point Number In `Environment/Debugger/Floating Point Format Display` – Custom	The printf format directive used to display floating point numbers.
Maximum Backtrace Calls `Environment/Debugger/Maximum Backtrace Calls` – IntegerRange	Selects the maximum number of calls when backtracing.

Prompt To Display If More Than `Environment/Debugger/Array Elements Prompt Size` – IntegerRange	The array size to display with prompt.
Show Data Tips In Text Editor `Environment/Debugger/Show Data Tips` – Boolean	Show Data Tips In Text Editor.
Show ELF Header `ElfDwarf/Environment/Show ELF Header` – Boolean	Display ELF Headers when executable and object files are displayed in text editor.
Show Folds In Disassembly `Environment/Debugger/Disassembly Show Folds` – Boolean	Show Folds In Disassembly.
Show Labels In Disassembly `Environment/Debugger/Disassembly Show Labels` – Boolean	Show Labels In Disassembly.
Show Source In Disassembly `Environment/Debugger/Disassembly Show Source` – Boolean	Show Source In Disassembly.
Show char * as null terminated string `Environment/Debugger/Display Char Ptr As String` – Boolean	Show char * as null terminated string.
Source Path `Environment/Debugger/Source Path` – StringList	Global search path to find source files.
Use objdump For File Disassembly `ElfDwarf/Environment/Use Objdump For Disassembly` – Boolean	Specifies whether to use objdump to disassemble files rather than the built-in disassembler.

Extended Data Tips

Property	**Description**
ASCII `Environment/Debugger/Extended Tooltip Display Mode/ASCII` – Boolean	Display ASCII extended data tips.
Binary `Environment/Debugger/Extended Tooltip Display Mode/Binary` – Boolean	Display Binary extended data tips.
Decimal `Environment/Debugger/Extended Tooltip Display Mode/Decimal` – Boolean	Display Decimal extended data tips.
Hexadecimal `Environment/Debugger/Extended Tooltip Display Mode/Hexadecimal` – Boolean	Display Hexadecimal extended data tips.

Octal `Environment/Debugger/Extended Tooltip Display Mode/Octal` – Boolean	Display Octal extended data tips.
Unsigned Decimal `Environment/Debugger/Extended Tooltip Display Mode/Unsigned Decimal` – Boolean	Display Unsigned Decimal extended data tips.

Ozone

Property	Description
Ozone Executable `ARM/Debugger/Ozone Executable` – FileName	The path to the Ozone executable.

Target

Property	Description
Switch Project To Text Editor `Environment/Debugger/Switch Project` – Boolean	Switch Debugger Project To Text Editor.

Target Log

Property	Description
Highlight Links `Environment/Target Log/Highlight Links` – Boolean	Highlight and navigate URL links in output.

Window

Property	Description
Clear Debug Terminal On Run `Environment/Clear Debug Terminal On Run` – Boolean	Clear the debug terminal automatically when a program is run.
Hide Output Window On Successful Load `Debugging/Hide Transcript On Successful Load` – Boolean	Hide the Output window when a load completes without error.
Show Target Log On Load `Debugging/Show Transcript On Load` – Boolean	Show the target log when a load starts.

IDE Environment Options

Browser

Property	Description
Text Size `Environment/Browser/Text Size` – Enumeration	Sets the text size of the integrated HTML and help browser.
Underline Hyperlinks In Browser `Environment/Browser/Underline Web Links` – Boolean	Enables underlining of hypertext links in the integrated HTML and help browser.

File Extension

Property	Description
ELF Archive File Extensions `ElfDwarf/Environment/Archive File Extensions` – StringList	The file extensions used for ELF archive files.
ELF Executable File Extensions `ElfDwarf/Environment/Executable File Extensions` – StringList	The file extensions used for ELF executable files.
ELF Object File Extensions `ElfDwarf/Environment/Object File Extensions` – StringList	The file extensions used for ELF object files.

File Search

Property	Description
Collapse Search Results `Find In Files/Collapse Results` – Boolean	Whether to initially collapse search results.
Files To Exclude `Find In Files/Exclude File Types` – StringList	The wildcard used to exclude files in Find In Files searches.
Files To Search `Find In Files/File Type` – StringList	The wildcard used to match files in Find In Files searches.
Find History `Find In Files/Find History` – StringList	The list of strings recently used in searches.
Flat Search Result Output `Find In Files/Flat Output` – Boolean	Whether to display file search results as a flat list.
Folder History `Find In Files/Folder History` – StringList	The set of folders recently used in file searches.

Match Case `Find In Files/Match Case` – Boolean	Whether the case of letters must match exactly when searching.
Match Whole Word `Find In Files/Match Whole Word` – Boolean	Whether the whole word must match when searching.
Replace History `Find In Files/Replace History` – StringList	The list of strings recently used in searches.
Search Dependencies `Find In Files/Search Dependencies` – Boolean	Controls searching of dependent files."
Search In `Find In Files/Context` – Enumeration	Where to look to find files.
Use Regular Expressions `Find In Files/Use RegExp` – Boolean	Whether to use a regular expression or plain text search.

Find And Replace

Property	Description
Greedy Regular Expressions `Find/Greedy RegExp` – Boolean	Enables greedy matching when using regular expressions.

Internet

Property	Description
Automatically Check For Packages `Environment/Internet/Check Packages` – Boolean	Specifies whether to enable downloading of the list of available packages.
Automatically Check For Updates `Environment/Internet/Check Updates` – Boolean	Specifies whether to check for software updates.
Check For Latest News `Environment/Internet/RSS Update` – Boolean	Specifies whether to update the latest news window.
Enable Connection Debugging `Environment/Internet/Enable Debugging` – Boolean	Controls debugging traces of internet connections and downloads.
External Web Browser `Environment/External Web Browser` – FileName	The path to the external web browser to use when accessing non-local files.
HTTP Caching `Environment/Internet/HTTP Caching` – Boolean	Specifies if caching should be permitted when carrying out HTTP requests.
HTTP Proxy Host `Environment/Internet/HTTP Proxy Server` – String	Specifies the IP address or hostname of the HTTP proxy server. If empty, no HTTP proxy server will be used.

HTTP Proxy Port `Environment/Internet/HTTP Proxy Port` – IntegerRange	Specifies the HTTP proxy server's port number.
Maximum Concurrent Downloads `Environment/Internet/Max Concurrent Downloads` – IntegerRange	The maximum number of downloads to carry out simultaneously.
Maximum Download History Items `Environment/Internet/Max Download History Items` – IntegerRange	The maximum amount of download history kept in the downloads window.

Language Server

Property	Description
Additional clangd Options `Environment/Language Server/Additional clangd Options` – StringList	Specifies additional command line options to use when starting clangd.
Enable Language Server Debugging `Environment/Language Server/Enable Debugging` – Boolean	Controls debugging traces of language servers.

Launcher

Property	Description
Confirm Check Solution Target `Environment/Launcher/Confirm Check Solution Target` – Boolean	Specifies whether the SEGGER Embedded Studio launcher should present a warning if the solution being launched targets a device it does not support.
Launch Latest Installations Only `Environment/Launcher Use Latest Installations Only` – Boolean	Specifies whether the SEGGER Embedded Studio launcher should only consider the latest installations when deciding which one to use.
Launcher Enabled `Environment/Launcher Enabled` – Boolean	Specifies whether the SEGGER Embedded Studio launcher should be used when the operating system or an external application requests a file to be opened.

Licensing

Property	Description
Check J-Link For Licenses `Environment/Check J-Link Licenses` – Boolean	Specifies whether to check J-Link's for licenses.

Package Manager

Property	Description
Check Solution Package Dependencies `Environment/Package/Check Solution Package Dependencies` – Boolean	Specifies whether to check package dependencies when a solution is loaded.
Delete Package Downloads `Environment/Package/Delete Downloads` – Boolean	Specifies whether to delete downloaded package files after they have been installed.
Install Default Packages `Environment/Package/Install Default Packages` – Boolean	Specifies whether default packages should be installed on startup if they are not installed already.
Package Directory `Environment/Package/Destination Directory` – String	Specifies the directory packages are installed to.
Parallel Download And Install `Environment/Package/Parallel Download And Install` – Boolean	Specifies whether the package manager should download and install packages in parallel.
Show Check For Packages Dialog `Environment/Package/Show Check For Packages Dialog` – Boolean	Specifies whether the package manager should prompt for a package list refresh.
Show Logos `Environment/Package/Show Logos` – Enumeration	Specifies whether the package manager should display company logos.
Verify Package Downloads `Environment/Package/Verify Downloads` – Boolean	Specifies whether to carry out an MD5 sum check on downloaded package files.

Print

Property	Description
Bottom Margin `Environment/Printing/Bottom Margin` – IntegerRange	The page's bottom margin in millimetres.
Left Margin `Environment/Printing/Left Margin` – IntegerRange	The page's left margin in millimetres.
Page Orientation `Environment/Printing/Orientation` – Enumeration	The page's orientation.
Page Size `Environment/Printing/Page Size` – Enumeration	The page's size.

Right Margin `Environment/Printing/Right Margin` – IntegerRange	The page's right margin in millimetres.
Top Margin `Environment/Printing/Top Margin` – IntegerRange	The page's top margin in millimetres.

References Search

Property	Description
Flat Search Result Output `Find References/Flat Output` – Boolean	Whether to display reference search results as a flat list.

Startup

Property	Description
Allow Multiple SEGGER Embedded Studios `Environment/Permit Multiple Studio Instances` – Boolean	Allow more than one SEGGER Embedded Studio to run at the same time.
Environment Variables `Environment/Environment Variables` – StringList	Environment variables to define on startup.
Load Last Project On Startup `Environment/Load Last Project On Startup` – Boolean	Specifies whether to load the last project the next time SEGGER Embedded Studio runs.
New Project Directory `Environment/General/Solution Directory` – String	The directory where projects are created.
Show Support Expired Dialog `Environment/Support Reminder Nag` – Boolean	Specifies whether to show the support expired dialog.
Sort Project File On Save `Environment/Sort Project File` – Boolean	The project file is sorted when it is saved.
Splash Screen `Environment/Splash Screen` – Enumeration	How to display the splash screen on startup.

Status Bar

Property	Description
(Visible) `Environment/Status Bar` – Boolean	Show or hide the status bar.

Show Build Status Pane `Environment/General/Status Bar/Show Build Status` – Boolean	Show or hide the Build pane in the status bar.
Show Caret Position Pane `Environment/General/Status Bar/Show Caret Pos` – Boolean	Show or hide the Caret Position pane in the status bar.
Show Insert/Overwrite Status Pane `Environment/General/Status Bar/Show Insert Mode` – Boolean	Show or hide the Insert/Overwrite pane in the status bar.
Show Read-Only Status Pane `Environment/General/Status Bar/Show Read Only` – Boolean	Show or hide the Read Only pane in the status bar.
Show Size Grip `Environment/General/Status Bar/Show Size Grip` – Boolean	Show or hide the status bar size grip.
Show Target Pane `Environment/General/Status Bar/Show Target` – Boolean	Show or hide the Target pane in the status bar.
Show Time Pane `Environment/General/Status Bar/Show Time` – Boolean	Show or hide the Time pane in the status bar.

Title Bar

Property	Description
Show Full Solution Path `Environment/General/Title Bar/Show Full Solution Path` – Boolean	Show the full solution path in title bar.

User Interface

Property	Description
Application Main Font `Environment/Application Main Font` – Font	The font to use for the user interface as a whole.
Application Monospace Font `Environment/Application Monospace Font` – FixedPitchFont	The fixed-size font to use for the user interface as a whole.
Default Document Markup Language Operation `Environment/Default Document Markup Language Operation` – Enumeration	The default operation to carry out when opening HTML and Markdown files.

Error Display Timeout `Environment/Error Display Timeout` – IntegerRange	The minimum time, in seconds, that errors are shown for in the status bar.
Errors Are Displayed `Environment/Error Display Mode` – Enumeration	How errors are reported in SEGGER Embedded Studio.
File Size Display Units `Environment/Size Display Unit` – Enumeration	How to display sizes of items in the user interface. SI defines 1kB=1000 bytes, IEC defines 1kiB=1024 bytes, Alternate SI defines 1kB=1024 bytes.
Number File Names in Menus `Environment/Number Menus` – Boolean	Number the first nine file names in menus for quick keyboard access.
Qt Style Sheet `Environment/Qt Style Sheet` – FileName	The Qt style sheet to use in order to customize the user interface (experimental).
Show Large Icons In Toolbars `Environment/General/Large Icons` – Enumeration	Show large or small icons on toolbars.
Show Ribbon `Environment/General/Ribbon/Show` – Boolean	Show or hide the ribbon.
Show Window Selector On Ctrl+Tab `Environment/Show Selector` – Boolean	Present the Window Selector on Next Window and Previous Window commands activated from the keyboard.
Theme `Environment/Studio Theme` – Enumeration	The user interface style and color theme to use.
Window Menu Contains At Most `Environment/Max Window Menu Items` – IntegerRange	The maximum number of windows appearing in the Windows menu.

Programming Language Environment Options

Assembly Language

Property	Description
Column Guide Columns `Text Editor/Indent/Assembly Language/Column Guides` – String	The columns that guides are drawn for.
Indent Mode `Text Editor/Indent/Assembly Language/Indent Mode` – Enumeration	How to indent when a new line is inserted.
Indent Size `Text Editor/Indent/Assembly Language/Size` – IntegerRange	The number of columns to indent a code block.
Tab Size `Text Editor/Indent/Assembly Language/Tab Size` – IntegerRange	The number of columns between tabstops.
Use Tabs `Text Editor/Indent/Assembly Language/Use Tabs` – Boolean	Insert tabs when indenting.
User-Defined Keywords `Text Editor/Indent/Assembly Language/Keywords` – StringList	Additional identifiers to highlight as keywords.

C and C++

Property	Description
Column Guide Columns `Text Editor/Indent/C and C++/Column Guides` – String	The columns that guides are drawn for.
Indent Mode `Text Editor/Indent/C and C++/Indent Mode` – Enumeration	How to indent when a new line is inserted.
Indent Size `Text Editor/Indent/C and C++/Size` – IntegerRange	The number of columns to indent a code block.
On Type Formatting `Text Editor/C and C++/Formatting/On Type Formatting` – Boolean	Use the code formatter to format the code as it is edited.

Tab Size `Text Editor/Indent/C and C++/Tab Size` – IntegerRange	The number of columns between tabstops.
Use Tabs `Text Editor/Indent/C and C++/Use Tabs` – Boolean	Insert tabs when indenting.
User-Defined Keywords `Text Editor/Indent/C and C++/Keywords` – StringList	Additional identifiers to highlight as keywords.

Default

Property	Description
Column Guide Columns `Text Editor/Indent/Default/Column Guides` – String	The columns that guides are drawn for.
Indent Mode `Text Editor/Indent/Default/Indent Mode` – Enumeration	How to indent when a new line is inserted.
Indent Size `Text Editor/Indent/Default/Size` – IntegerRange	The number of columns to indent a code block.
Tab Size `Text Editor/Indent/Default/Tab Size` – IntegerRange	The number of columns between tabstops.
Use Tabs `Text Editor/Indent/Default/Use Tabs` – Boolean	Insert tabs when indenting.
User-Defined Keywords `Text Editor/Indent/Default/Keywords` – StringList	Additional identifiers to highlight as keywords.

JSON

Property	Description
Column Guide Columns `Text Editor/Indent/JSON/Column Guides` – String	The columns that guides are drawn for.
Indent Mode `Text Editor/Indent/JSON/Indent Mode` – Enumeration	How to indent when a new line is inserted.

Indent Size `Text Editor/Indent/JSON/Size` – IntegerRange	The number of columns to indent a code block.
Tab Size `Text Editor/Indent/JSON/Tab Size` – IntegerRange	The number of columns between tabstops.
Use Tabs `Text Editor/Indent/JSON/Use Tabs` – Boolean	Insert tabs when indenting.
User-Defined Keywords `Text Editor/Indent/JSON/Keywords` – StringList	Additional identifiers to highlight as keywords.

Java

Property	Description
Column Guide Columns `Text Editor/Indent/Java/Column Guides` – String	The columns that guides are drawn for.
Indent Mode `Text Editor/Indent/Java/Indent Mode` – Enumeration	How to indent when a new line is inserted.
Indent Size `Text Editor/Indent/Java/Size` – IntegerRange	The number of columns to indent a code block.
Tab Size `Text Editor/Indent/Java/Tab Size` – IntegerRange	The number of columns between tabstops.
Use Tabs `Text Editor/Indent/Java/Use Tabs` – Boolean	Insert tabs when indenting.
User-Defined Keywords `Text Editor/Indent/Java/Keywords` – StringList	Additional identifiers to highlight as keywords.

JavaScript

Property	Description
Column Guide Columns `Text Editor/Indent/JavaScript/Column Guides` – String	The columns that guides are drawn for.
Indent Mode `Text Editor/Indent/JavaScript/Indent Mode` – Enumeration	How to indent when a new line is inserted.
Indent Size `Text Editor/Indent/JavaScript/ Size` – IntegerRange	The number of columns to indent a code block.

Tab Size `Text Editor/Indent/JavaScript/Tab Size` – IntegerRange	The number of columns between tabstops.
Use Tabs `Text Editor/Indent/JavaScript/Use Tabs` – Boolean	Insert tabs when indenting.
User-Defined Keywords `Text Editor/Indent/JavaScript/Keywords` – StringList	Additional identifiers to highlight as keywords.

XML

Property	Description
Column Guide Columns `Text Editor/Indent/XML/Column Guides` – String	The columns that guides are drawn for.
Indent Mode `Text Editor/Indent/XML/Indent Mode` – Enumeration	How to indent when a new line is inserted.
Indent Size `Text Editor/Indent/XML/Size` – IntegerRange	The number of columns to indent a code block.
Tab Size `Text Editor/Indent/XML/Tab Size` – IntegerRange	The number of columns between tabstops.
Use Tabs `Text Editor/Indent/XML/Use Tabs` – Boolean	Insert tabs when indenting.
User-Defined Keywords `Text Editor/Indent/XML/Keywords` – StringList	Additional identifiers to highlight as keywords.

Source Control Environment Options

External Tools

Property	Description
Diff Command Line `Environment/Source Code Control/DiffCommand` – StringList	The diff command line.
Merge Command Line `Environment/Source Code Control/MergeCommand` – StringList	The merge command line.

Preference

Property	Description
Add Immediately `Environment/Source Code Control/ImmediateAdd` – Boolean	Bypasses the confirmation dialog and immediately adds items to source control.
Commit Immediately `Environment/Source Code Control/ImmediateCommit` – Boolean	Bypasses the confirmation dialog and immediately commits items.
Get Immediately `Environment/Source Code Control/ImmediateGet` – Boolean	Bypasses the confirmation dialog and immediately gets items from source control.
Lock Immediately `Environment/Source Code Control/ImmediateLock` – Boolean	Bypasses the confirmation dialog and immediately locks items.
Remove Immediately `Environment/Source Code Control/ImmediateRemove` – Boolean	Bypasses the confirmation dialog and immediately removes items from source control.
Resolved Immediately `Environment/Source Code Control/ImmediateResolved` – Boolean	Bypasses the confirmation dialog and immediately mark items resolved.
Revert Immediately `Environment/Source Code Control/ImmediateRevert` – Boolean	Bypasses the confirmation dialog and immediately revert items.
Unlock Immediately `Environment/Source Code Control/ImmediateUnlock` – Boolean	Bypasses the confirmation dialog and immediately unlocks items.

Update Immediately `Environment/Source Code Control/Immediate Update` – Boolean	Bypasses the confirmation dialog and immediately updates items.

Text Editor Environment Options

Auto Recovery

Property	Description
Auto Recovery Backup Time `Text Editor/Auto Recovery Backup Time` – IntegerRange	The time in minutes between saving of auto recovery backups files or 0 to disable generation of backup files.
Auto Recovery Keep Time `Text Editor/Auto Recovery Keep Time` – IntegerRange	The time in days to keep unrecovered backup files or 0 to disable deletion of unrecovered backup files.

Cursor Fence

Property	Description
Bottom Margin `Text Editor/Margins/Bottom` – IntegerRange	The number of lines in the bottom margin.
Keep Cursor Within Fence `Text Editor/Margins/Enabled` – Boolean	Enable margins to fence and scroll around the cursor.
Left Margin `Text Editor/Margins/Left` – IntegerRange	The number of characters in the left margin.
Right Margin `Text Editor/Margins/Right` – IntegerRange	The number of characters in the right margin.
Top Margin `Text Editor/Margins/Top` – IntegerRange	The number of lines in the right margin.

Editing

Property	Description
Allow Drag and Drop Editing `Text Editor/Drag Drop Editing` – Boolean	Enables dragging and dropping of selections in the text editor.
Bold Popup Diagnostic Messages `Text Editor/Bold Popup Diagnostics` – Boolean	Displays popup diagnostic messages in bold for easier reading.
Column-mode Tab `Text Editor/Column Mode Tab` – Boolean	Tab key moves to the next textual column using the line above.
Confirm Modified File Reload `Text Editor/Confirm Modified File Reload` – Boolean	Display a confirmation prompt before reloading a file that has been modified on disk.

Copy Action When Nothing Selected `Text Editor/Copy Action` – Enumeration	What Copy copies when nothing is selected.
Cut Action When Nothing Selected `Text Editor/Cut Action` – Enumeration	What Cut cuts when nothing is selected.
Cut Single Blank Line `Text Editor/Cut Blank Lines` – Boolean	Selects whether to place text on the clipboard when a single blank line is cut. When set to Yes, cutting a single blank line will put the blank line on the clipboard. When set to No, cutting a single blank line deletes the line and does not place it on the clipboard.
Diagnostic Cycle Mode `Text Editor/Diagnostic Cycle Mode` – Enumeration	Iterates through diagnostics either from most severe to least severe or in reported order.
Edit Read-Only Files `Text Editor/Edit Read Only` – Boolean	Allow editing of read-only files.
Enable Virtual Space `Text Editor/Enable Virtual Space` – Boolean	Permit the cursor to move into locations that do not currently contain text.
Numeric Keypad Editing `Text Editor/Numeric Keypad Enabled` – Boolean	Selects whether the numeric keypad plus and minus buttons copy and cut text.
Tab Key Indents Preprocessor Directives `Text Editor/Tab Key Indents Preprocessor Directives` – Boolean	Enables or disables the indentation of C preprocessor directives when using tab key indentation on a selection.
Undo And Redo Behavior `Text Editor/Undo Mode` – Enumeration	How Undo and Redo group your typing when it is undone and redone.

Find And Replace

Property	Description
Case Sensitive Matching `Text Editor/Find/Match Case` – Boolean	Enables or disables the case sensitivity of letters when searching.
Find History `Text Editor/Find/History` – StringList	The list of strings recently used in searches.
Regular Expression Matching `Text Editor/Find/Use RegExp` – Boolean	Enables regular expression matching rather than plain text matching.
Replace History `Text Editor/Replace/History` – StringList	The list of strings recently used in replaces.
Whole Word Matching `Text Editor/Find/Match Whole Word` – Boolean	Enables or disables whole word matching when searching.

Formatting

Property	Description
Formatting Options `Text Editor/Formatting Options` – StringList	Specify Clang-Format options. See https://clang.llvm.org/docs/ClangFormatStyleOptions.html for a description of the Clang-Format options available.
Formatting Style `Text Editor/Formatting Style` – Enumeration	Select a set of formatting options based on a named standard.
Use .clang-format File `Text Editor/Use .clang-format File` – Boolean	Load code formatting style configuration from a .clang-format file located in one of the parent directories of the source file rather than use the formatting options.

International

Property	Description
Auto-Detect UTF-8 `Text Editor/Auto-Detect UTF-8` – Boolean	Auto-detect UTF-8 encoding without signature.
Default Text File Encoding `Text Editor/Default Codec` – Enumeration	The encoding to use if not overridden by a project property or file is not in a known format.
Verify Text File Decoding `Text Editor/Verify Decode` – Boolean	Specifies whether the decoding of a text file should be verified when file is loaded.

Mouse

Property	Description
Alt+Left Click Action `Environment/Project Explorer/Alt+Left Click Action` – Enumeration	The action the editor performs on Alt+Left Click.
Alt+Middle Click Action `Environment/Project Explorer/Alt+Middle Click Action` – Enumeration	The action the editor performs on Alt+Middle Click.
Alt+Right Click Action `Environment/Project Explorer/Alt+Right Click Action` – Enumeration	The action the editor performs on Alt+Right Click.
Copy On Mouse Select `Text Editor/Copy On Mouse Select` – Boolean	Automatically copy text to clipboard when marking a selection with the mouse.
Ctrl+Left Click Action `Environment/Project Explorer/Ctrl+Left Click Action` – Enumeration	The action the editor performs on Ctrl+Left Click.

Ctrl+Middle Click Action `Environment/Project Explorer/Ctrl+Middle Click Action` – Enumeration	The action the editor performs on Ctrl+Middle Click.
Ctrl+Right Click Action `Environment/Project Explorer/Ctrl+Right Click Action` – Enumeration	The action the editor performs on Ctrl+Right Click.
Middle Click Action `Environment/Project Explorer/Middle Click Action` – Enumeration	The action the editor performs on Middle Click.
Mouse Wheel Adjusts Font Size `Text Editor/Mouse Wheel Adjusts Font Size` – Boolean	Enable or disable resizing of font by mouse wheel when CTRL key pressed.
Shift+Middle Click Action `Environment/Project Explorer/Shift+Middle Click Action` – Enumeration	The action the editor performs on Shift+Middle Click.
Shift+Right Click Action `Environment/Project Explorer/Shift+Right Click Action` – Enumeration	The action the editor performs on Shift+Right Click.

Programmer Assistance

Property	**Description**
ATTENTION Tag List `Text Editor/ATTENTION Tags` – StringList	Set the tags to display as ATTENTION comments.
Ask For Index `Text Editor/Ask For Index` – Boolean	Ask to index the project if goto symbol fails in current editor context.
Auto-Comment Text `Text Editor/Auto Comment` – Boolean	Enable or disable automatically swapping commenting on source lines by typing '/' with an active selection.
Auto-Surround Text `Text Editor/Auto Surround` – Boolean	Enable or disable automatically surrounding selected text when typing triangular brackets, quotation marks, parentheses, brackets, or braces.
Check Spelling `Text Editor/Spell Checking` – Boolean	Enable spell checking in comments.
Code Completion Characters `Text Editor/Code Completion Characters` – IntegerRange	The minimum number of word characters required before showing the code completion suggestions while typing.
Code Completion Replaces Existing Word `Text Editor/Completion Replaces Existing Word` – Boolean	Replace existing word with completion suggestion if cursor is located on one.

Code Completion Suggestion Selection Key `Text Editor/Suggestion Selection Key` – Enumeration	The key used to select a code completion suggestion.
Code Completion Within Word `Text Editor/Code Completion Within Word` – Boolean	Specifies whether the cursor can be located within a word when showing the code completion suggestions while typing.
Display Code Completion Suggestions While Typing `Text Editor/Suggest Completion While Typing` – Boolean	Enable code completion as you type without needing to use the show suggestions key (Ctrl+J).
Enable Popup Diagnostics `Text Editor/Enable Popup Diagnostics` – Boolean	Enables on-screen popup diagnostics messages.
FIXME Tag List `Text Editor/FIXME Tags` – StringList	Set the tags to display as FIXME comments.
Inactive Code Opacity `Text Editor/Inactive Code Opacity` – IntegerRange	Specifies the opacity of code that has been conditionally excluded by the preprocessor.
Include Preprocessor Definitions in Suggestions `Text Editor/Preprocessor Definition Suggestions` – Boolean	Include or exclude preprocessor definitions in code completion suggestions.
Include Templates in Suggestions `Text Editor/Template Suggestions` – Boolean	Include or exclude templates in code completion suggestions.
Lint Tag List `Text Editor/LINT Tags` – StringList	Set the tags to display as Lint directives.
Select First Code Completion Selection `Text Editor/Select First Suggestion` – Boolean	Automatically select first suggestion when showing suggestions
Show Diagnostics `Text Editor/Show Diagnostics` – Enumeration	Enables on-screen diagnostics in the text editor.
Show Inactive Code `Text Editor/Show Inactive Code` – Enumeration	Show code that has been conditionally excluded by the preprocessor.
Show Inline Diagnostics `Text Editor/Show Inline Diagnostics` – Enumeration	Enables inline diagnostics in the text editor.
Show Symbol Declaration Tooltips `Text Editor/Show Tooltip` – Boolean	Show tooltips when hovering over symbols.
Template Characters To Match `Text Editor/Template Suggestions Characters` – IntegerRange	The number of characters to match before suggesting a template.

Save

Property	Description
Backup File History Depth `Text Editor/Backup File Depth` – IntegerRange	The number of backup files to keep when saving an existing file.
Default Line Endings `Text Editor/Default Line Endings` – Enumeration	The line ending format to use for a new file or a file where the existing line ending format cannot be determined.
Delete Trailing Space On Save `Text Editor/Delete Trailing Space On Save` – Boolean	Deletes trailing whitespace from each line when a file is saved.
Format On Save `Text Editor/Format On Save` – Enumeration	Formats text when a file is saved.
Tab Cleanup On Save `Text Editor/Cleanup Tabs On Save` – Enumeration	Cleans up tabs when a file is saved.

Visual Appearance

Property	Description
Fold Comments `Text Editor/Fold Comments` – Boolean	Allow multiline comments to be collapsed.
Fold Preprocessor Directives `Text Editor/Fold Preprocessor Directives` – Boolean	Allow preprocessor directives to be collapsed.
Font `Text Editor/Font` – FixedPitchFont	The font to use for text editors.
Font Rendering `Text Editor/Font Rendering` – Enumeration	The font rendering scheme to use in text editors.
Font Smoothing Threshold `Text Editor/Antialias Threshold` – IntegerRange	The minimum size for font smoothing: font sizes smaller than this will have antialiasing turned off.
Hide Cursor When Typing `Text Editor/Hide Cursor When Typing` – Boolean	Hide or show the I-beam cursor when you start to type.
Highlight All Selected Text `Text Editor/Highlight All Selected Text` – Boolean	Enable or disable visually highlighting all text that matches the current selection.
Highlight Cursor Line `Text Editor/Highlight Cursor Line` – Boolean	Enable or disable visually highlighting the cursor line.

Highlight References `Text Editor/Highlight References` – Boolean	Enable or disable visually highlighting of references.
Horizontal Scroll Bar `Text Editor/HScroll Bar` – Enumeration	Show or hide the horizontal scroll bar.
Insert Caret Style `Text Editor/Insert Caret Style` – Enumeration	How the caret is displayed with the editor in insert mode.
Line Numbers `Text Editor/Line Number Mode` – Enumeration	How often line numbers are displayed in the margin.
Mate Match Off Screen `Text Editor/Mate Match Off Screen` – Boolean	Specifies whether braces, brackets, and parentheses are matched when off screen.
Mate Matching Mode `Text Editor/Mate Matching Mode` – Enumeration	Controls when braces, brackets, and parentheses are matched.
Maximum Collapsed Fold Preview Lines `Text Editor/Maximum Collapsed Fold Preview Lines` – IntegerRange	The maximum number of lines to show in a collapsed fold preview tooltip.
Minimum Scroll Width `Text Editor/Minimum Scroll Width` – IntegerRange	Specifies the minimum width of the scrolling region in characters.
Overwrite Caret Style `Text Editor/Overwrite Caret Style` – Enumeration	How the caret is displayed with the editor in overwrite mode.
Selection Opacity `Text Editor/Selection Opacity` – IntegerRange	Specifies the opacity of text selection.
Show Bookmarks In Vertical Scroll Bar `Text Editor/Show Bookmarks In Vertical Scroll Bar` – Boolean	Annotate the vertical scroll bar with bookmark positions.
Show Breakpoints In Vertical Scroll Bar `Text Editor/Show Breakpoints In Vertical Scroll Bar` – Boolean	Annotate the vertical scroll bar with breakpoint positions.
Show Caret Position In Vertical Scroll Bar `Text Editor/Show Caret In Vertical Scroll Bar` – Boolean	Annotate the vertical scroll bar with the caret's position within the document.
Show Diagnostic Icons In Gutter `Text Editor/Diagnostic Icons` – Boolean	Enables display of diagnostic icons in the icon gutter.
Show Errors In Vertical Scroll Bar `Text Editor/Show Errors In Vertical Scroll Bar` – Boolean	Annotate the vertical scroll bar with error positions.
Show Fold Gutter `Text Editor/Fold Gutter` – Boolean	Show or hide the left-hand gutter containing folding controls.
Show Icon Gutter `Text Editor/Icon Gutter` – Boolean	Show or hide the left-hand gutter containing breakpoint, bookmark, and optional diagnostic icons.

Show Mini Toolbar `Text Editor/Mini Toolbar` – Boolean	Show the mini toolbar when selecting text with the mouse.
Show Toolbar `Text Editor/ShowWidgetStrip` – Boolean	Show or hide the Editor toolbar in the dock window.
Show Warnings In Vertical Scroll Bar `Text Editor/Show Warnings In Vertical Scroll Bar` – Boolean	Annotate the vertical scroll bar with warning positions.
Use I-beam Cursor `Text Editor/Ibeam cursor` – Boolean	Show an I-beam or arrow cursor in the text editor.
Vertical Scroll Bar `Text Editor/VScroll Bar` – Enumeration	Show or hide the vertical scroll bar.
View Whitespace `Text Editor/View Whitespace` – Boolean	Make whitespace characters visible in the text editor.

Windows Environment Options

Autos

Property	Description
Show Digit Separator `Environment/AutosWindow/Show Digit Separator` – Boolean	Show digit separator in variable value display.
Show Member Functions `Environment/AutosWindow/Show Member Functions` – Boolean	Controls whether C++ class member functions are displayed.
Show Variable Address Column `Environment/AutosWindow/Show Address Column` – Boolean	Controls whether the variable address column is displayed.
Show Variable Size Column `Environment/AutosWindow/Show Size Column` – Boolean	Controls whether the variable size column is displayed.
Show Variable Type Column `Environment/AutosWindow/Show Type Column` – Boolean	Controls whether the variable type column is displayed.

Call Stack

Property	Description
Execution Frame at Top `Environment/Call Stack/Most Recent At Top` – Boolean	Controls whether the most recent call is at the top or the bottom of the list.
Show Call Address `Environment/Call Stack/Show Call Address` – Boolean	Enables the display of the call address in the call stack.
Show Call Source Location `Environment/Call Stack/Show Call Location` – Boolean	Enables the display of the call source location in the call stack.
Show Frame Size `Environment/Call Stack/Show Stack Usage` – Boolean	Enables the display of the amount of stack used by the call.
Show Frame Size In Bytes `Environment/Call Stack/Show Stack Usage In Bytes` – Boolean	Display the stack usage in bytes rather than words.

Show Parameter Names `Environment/Call Stack/Show Parameter Names` – Boolean	Enables the display of parameter names in the call stack.
Show Parameter Types `Environment/Call Stack/Show Parameter Types` – Boolean	Enables the display of parameter types in the call stack.
Show Parameter Values `Environment/Call Stack/Show Parameter Values` – Boolean	Enables the display of parameter values in the call stack.
Show Stack Pointer `Environment/Call Stack/Show Stack Pointer` – Boolean	Enables the display of the stack pointer in the call stack.
Show Stack Usage `Environment/Call Stack/Show Cumulative Stack Usage` – Boolean	Enables the display of the amount of stack used.
Show Stack Usage In Bytes `Environment/Call Stack/Show Cumulative Stack Usage In Bytes` – Boolean	Display the stack usage in bytes rather than words.

Clipboard Ring

Property	Description
Maximum Items Held In Ring `Environment/Clipboard Ring/Max Entries` – IntegerRange	The maximum number of items held on the clipboard ring before they are recycled.
Preserve Contents Between Runs `Environment/Clipboard Ring/Save` – Boolean	Save the clipboard ring across SEGGER Embedded Studio runs.

Debug Terminal

Property	Description
Backscroll Buffer Lines `Debug Terminal/Backscroll Buffer Lines` – IntegerRange	The number of lines you can see when you scroll backward in the debug terminal window.
Highlight Links `Debug Terminal/Highlight Links` – Boolean	Highlight and navigate URL links in output.
Use Window System Colors `Debug Terminal/Use Window System Colors` – Boolean	Substitute window system colors for ANSI black background and white foreground in debug terminal.

Find Symbol Dialog

Property	Description
Group Symbols `Windows/Find Symbol Dialog/Group Symbols` – Boolean	Group symbols by type.
Scope `Windows/Find Symbol Dialog/Scope` – Enumeration	Specifies whether to search for symbols in the entire workspace or only the current document.

Frame Buffer

Property	Description
Maximum Frame Buffer Height `Environment/Frame Buffer Window/Maximum Height` – IntegerRange	Specifies the maximum frame buffer height.
Maximum Frame Buffer Width `Environment/Frame Buffer Window/Maximum Width` – IntegerRange	Specifies the maximum frame buffer width.
Show Frame Buffer Tooltips `Environment/Frame Buffer Window/Display Tooltips` – Boolean	Specifies whether tooltips are displayed in the frame buffer window.

Globals

Property	Description
Show Digit Separator `Environment/GlobalsWindow/Show Digit Separator` – Boolean	Show digit separator in variable value display.
Show Member Functions `Environment/GlobalsWindow/Show Member Functions` – Boolean	Controls whether C++ class member functions are displayed.
Show Variable Address Column `Environment/GlobalsWindow/Show Address Column` – Boolean	Controls whether the variable address column is displayed.
Show Variable Size Column `Environment/GlobalsWindow/Show Size Column` – Boolean	Controls whether the variable size column is displayed.
Show Variable Type Column `Environment/GlobalsWindow/Show Type Column` – Boolean	Controls whether the variable type column is displayed.

Locals

Property	Description
Show Digit Separator `Environment/LocalsWindow/Show Digit Separator` – Boolean	Show digit separator in variable value display.
Show Member Functions `Environment/LocalsWindow/Show Member Functions` – Boolean	Controls whether C++ class member functions are displayed.
Show Struct Offsets `Environment/Watch4Window/Show Struct Offsets` – Boolean	Show offsets of structure fields in the address column.
Show Struct Offsets `Environment/Watch3Window/Show Struct Offsets` – Boolean	Show offsets of structure fields in the address column.
Show Struct Offsets `Environment/Watch2Window/Show Struct Offsets` – Boolean	Show offsets of structure fields in the address column.
Show Struct Offsets `Environment/Watch1Window/Show Struct Offsets` – Boolean	Show offsets of structure fields in the address column.
Show Struct Offsets `Environment/AutosWindow/Show Struct Offsets` – Boolean	Show offsets of structure fields in the address column.
Show Struct Offsets `Environment/GlobalsWindow/Show Struct Offsets` – Boolean	Show offsets of structure fields in the address column.
Show Struct Offsets `Environment/LocalsWindow/Show Struct Offsets` – Boolean	Show offsets of structure fields in the address column.
Show Variable Address Column `Environment/LocalsWindow/Show Address Column` – Boolean	Controls whether the variable address column is displayed.
Show Variable Size Column `Environment/LocalsWindow/Show Size Column` – Boolean	Controls whether the variable size column is displayed.
Show Variable Type Column `Environment/LocalsWindow/Show Type Column` – Boolean	Controls whether the variable type column is displayed.

Memory

Property	Description
Confirm Large Download `Environment/Memory Window/Confirm Size` – Boolean	Present a warning if you attempt to download a large amount of memory in the memory window.
Group Auto Columns `Environment/Memory Window/Group Auto Columns` – Boolean	Selects whether columns are grouped in automatic column mode.
Locate Sets Entry Width `Environment/Memory Window/Locate Sets Entry Width` – Boolean	Set the memory window entry width if possible when locating.
Locate Sets Size `Environment/Memory Window/Locate Sets Size` – Boolean	Set the memory window size when locating.
Scroll Wheel Modifies Start Address `Environment/Memory Window/Scroll Wheel Modifies Start Address` – Boolean	Selects whether the mouse scroll wheel can change the memory window start address.
Selecting Modifies Start Address `Environment/Memory Window/Selecting Modifies Start Address` – Boolean	Selects whether the memory window start address can change when selecting.

Outline

Property	Description
Group Top-Level Declarations `Windows/Outline/Group Top Level Items` – Boolean	Group consecutive top-level variable and type declarations.
Show Function Arguments `Windows/Outline/Show Function Args` – Boolean	Show function arguments.

Project Explorer

Property	Description
Add Filename Replace Macros `Environment/Project Explorer/Filename Replace Macros` – StringList	Macros (system and global) used to replace the start of a filename on project file addition.
Check Solution Target `Environment/Project Explorer/Check Solution Target` – Boolean	Specifies whether to check target is correct when loading a solution.

Color Project Nodes `Environment/Project Explorer/Color Nodes` – Boolean	Show the project nodes colored for identification in the Project Explorer.
Confirm Configuration Folder Delete `Project Explorer/Confirm Configuration Folder Delete` – Boolean	Display a confirmation prompt before deleting a configuration folder cotaining properties.
Confirm File Replacement Warning `Project Explorer/Confirm File Replacement Warning` – Boolean	Display a confirmation prompt before replacing project files for import and creation
Confirm Forget Modified Options `Project Explorer/Confirm Reject Property Changes` – Boolean	Display a confirmation prompt before forgetting option modifications.
Context Menu Edit Options At Top `Environment/Project Explorer/Context Menu Properties Position` – Boolean	Controls where Edit Options are displayed by the Project Explorer's context menu.
Context Menu Uses Common Folder `Environment/Project Explorer/Context Menu Common Folder` – Boolean	Controls how common options are displayed by the Project Explorer's context menu.
External Editor `Environment/Project Explorer/External Editor` – FileName	The file name of the application to use as the external text editor. The external editor is started by holding down the Shift key when opening files from the project explorer.
Highlight Dynamic Items `Environment/Project Explorer/Show Dynamic Overlay` – Boolean	Show an overlay on an item if it is populated from a dynamic folder.
Highlight External Items `Environment/Project Explorer/Show Non-Local Overlay` – Boolean	Show an overlay on an item if it is not held within the project directory.
Output Files Folder `Environment/Project Explorer/Show Output Files` – Boolean	Show the build output files in an Output Files folder in the project explorer.
Read-Only Data In Code `Environment/Project Explorer/Statistics Read-Only Data Handling` – Boolean	Configures whether read-only data contributes to the Code or Data statistic.
Show Dependencies `Environment/Project Explorer/Dependencies Display` – Enumeration	Controls how the dependencies are displayed.
Show Favorite Properties `Environment/Project Explorer/Context Menu Show Favorites` – Boolean	Controls if favorite properties are displayed by the Project Explorer's context menu.
Show File Count on Folder `Environment/Project Explorer/Count Files` – Boolean	Show the number of files contained in a folder as a badge in the Project Explorer.

Show Modified Options on Folder/File `Environment/Project Explorer/Show Modified Properties` – Boolean	Show if a folder or file has modified options as a badge in the Project Explorer.
Show Options `Environment/Project Explorer/Properties Display` – Enumeration	Controls how the options are displayed.
Show Project Count on Solution `Environment/Project Explorer/Count Projects` – Boolean	Show the number of projects contained in a solution as a badge in the Project Explorer.
Show Source Control Annotation `Environment/Project Explorer/Show Source Control Annotation` – Boolean	Annotate items in the project explorer with their source control status.
Show Statistics Rounded `Environment/Project Explorer/Statistics Format` – Boolean	Show exact or rounded sizes in the project explorer.
Source Control Status Column `Environment/Project Explorer/Show Source Control Column` – Boolean	Show the source control status column in the project explorer.
Starred Files Names `Environment/Project Explorer/Starred File Names` – StringList	The list of wildcard-matched file names that are highligted with stars, to bring attention to themselves, in the Project Explorer.
Statistics Column `Environment/Project Explorer/Statistics Display` – Boolean	Show the code and data size columns in the Project Explorer.
Synchronize Explorer With Editor `Environment/Project Explorer/Sync Editor` – Boolean	Synchronizes the Project Explorer with the document being edited.
Use Common Options Folder `Environment/Project Explorer/Common Properties Display` – Boolean	Controls how common options are displayed.

Registers 1

Property	Description
Show Digit Separator `Environment/Registers1Window/Show Digit Separator` – Boolean	Show digit separator in register value display.
Show Register Address Column `Environment/Registers1Window/Show Address Column` – Boolean	Controls whether the register address column is displayed.

Registers 2

Property	Description
Show Digit Separator `Environment/Registers2Window/Show Digit Separator` – Boolean	Show digit separator in register value display.
Show Register Address Column `Environment/Registers2Window/Show Address Column` – Boolean	Controls whether the register address column is displayed.

Registers 3

Property	Description
Show Digit Separator `Environment/Registers3Window/Show Digit Separator` – Boolean	Show digit separator in register value display.
Show Register Address Column `Environment/Registers3Window/Show Address Column` – Boolean	Controls whether the register address column is displayed.

Registers 4

Property	Description
Show Digit Separator `Environment/Registers4Window/Show Digit Separator` – Boolean	Show digit separator in register value display.
Show Register Address Column `Environment/Registers4Window/Show Address Column` – Boolean	Controls whether the register address column is displayed.

Source Navigator

Property	Description
Group Symbols `Windows/Source Navigator/Group Symbols` – Boolean	Group symbols by type.
Scope `Windows/Source Navigator/Scope` – Enumeration	Specifies whether to search for symbols in the entire workspace or only the current document.

Stack Usage

Property	Description
Ignore Symbols `Environment/Stack Usage/Ignore Symbols` – StringList	List of symbols to ignore when populating the stack usage window.

Symbol Browser

Property	Description
Code Field `Environment/Symbol Browser/Display Code` – Boolean	Selects whether the Code field is displayed.
Const Field `Environment/Symbol Browser/Display Const` – Boolean	Selects whether the Const field is displayed.
Data Field `Environment/Symbol Browser/Display Data` – Boolean	Selects whether the Data field is displayed.
Frame Size Field `Environment/Symbol Browser/Display Frame Size` – Boolean	Selects whether the Frame Size field is displayed.
Range Field `Environment/Symbol Browser/Display Range` – Boolean	Selects whether the Range field is displayed.
Section Field `Environment/Symbol Browser/Display Section` – Boolean	Selects whether the Section field is displayed.
Size Field `Environment/Symbol Browser/Display Size` – Boolean	Selects whether the Size field is displayed.
Sort Criteria `Environment/Symbol Browser/Grouping` – Enumeration	Selects how to sort or group the symbols displayed.
Type Field `Environment/Symbol Browser/Display Type` – Boolean	Selects whether the Type field is displayed.
Value Field `Environment/Symbol Browser/Display Value` – Boolean	Selects whether the Value field is displayed.

Terminal Emulator

Property	Description
Backscroll Buffer Lines `Terminal Emulator/Backscroll Buffer Lines` – IntegerRange	The number of lines you can see when you scroll backward in the terminal emulator window.
Baud Rate `Terminal Emulator/Communications/Baud Rate` – Enumeration	Baud rate used when transmitting and receiving data.
Data Bits `Terminal Emulator/Communications/Data Bits` – Enumeration	Number of data bits to use when transmitting and receiving data.
Flow Control `Terminal Emulator/Communications/Flow Control` – Enumeration	The flow control method to use.
Highlight Links `Terminal Emulator/Highlight Links` – Boolean	Highlight and navigate URL links in output.
Line Feed On Carriage Return `Terminal Emulator/Line Feed On Carriage Return` – Boolean	Append a line feed character when a carriage return character is received.
Local Echo `Terminal Emulator/Local Echo` – Boolean	Displays every character typed before sending to the remote computer.
Maximum Input Block Size `Terminal Emulator/Maximum Input Block Size` – IntegerRange	The maximum number of bytes to read at a time.
Parity `Terminal Emulator/Communications/Parity` – Enumeration	Parity used when transmitting and receiving data.
Port `Terminal Emulator/Communications/Port` – COMPort	The communications port to use, e.g. /dev/ttyS0, /dev/ttyS1, etc.
Port Used By Target Interface `Terminal Emulator/Communications/Port Used By Target Interface` – Boolean	The COM port will be disconnected when the target interface is connected and reconnected when the target interface is disconnected.
Set DTR `Terminal Emulator/Communications/DTR` – Boolean	Set the DTR signal.
Stop Bits `Terminal Emulator/Communications/Stop Bits` – Enumeration	Number of stop bits to use when transmitting data.

Watch 1

Property	Description
Show Digit Separator `Environment/Watch1Window/Show Digit Separator` – Boolean	Show digit separator in variable value display.
Show Member Functions `Environment/Watch1Window/Show Member Functions` – Boolean	Controls whether C++ class member functions are displayed.
Show Variable Address Column `Environment/Watch1Window/Show Address Column` – Boolean	Controls whether the variable address column is displayed.
Show Variable Size Column `Environment/Watch1Window/Show Size Column` – Boolean	Controls whether the variable size column is displayed.
Show Variable Type Column `Environment/Watch1Window/Show Type Column` – Boolean	Controls whether the variable type column is displayed.

Watch 2

Property	Description
Show Digit Separator `Environment/Watch2Window/Show Digit Separator` – Boolean	Show digit separator in variable value display.
Show Member Functions `Environment/Watch2Window/Show Member Functions` – Boolean	Controls whether C++ class member functions are displayed.
Show Variable Address Column `Environment/Watch2Window/Show Address Column` – Boolean	Controls whether the variable address column is displayed.
Show Variable Size Column `Environment/Watch2Window/Show Size Column` – Boolean	Controls whether the variable size column is displayed.
Show Variable Type Column `Environment/Watch2Window/Show Type Column` – Boolean	Controls whether the variable type column is displayed.

Watch 3

Property	Description

Show Digit Separator `Environment/Watch3Window/Show Digit Separator` – Boolean	Show digit separator in variable value display.
Show Member Functions `Environment/Watch3Window/Show Member Functions` – Boolean	Controls whether C++ class member functions are displayed.
Show Variable Address Column `Environment/Watch3Window/Show Address Column` – Boolean	Controls whether the variable address column is displayed.
Show Variable Size Column `Environment/Watch3Window/Show Size Column` – Boolean	Controls whether the variable size column is displayed.
Show Variable Type Column `Environment/Watch3Window/Show Type Column` – Boolean	Controls whether the variable type column is displayed.

Watch 4

Property	Description
Show Digit Separator `Environment/Watch4Window/Show Digit Separator` – Boolean	Show digit separator in variable value display.
Show Member Functions `Environment/Watch4Window/Show Member Functions` – Boolean	Controls whether C++ class member functions are displayed.
Show Variable Address Column `Environment/Watch4Window/Show Address Column` – Boolean	Controls whether the variable address column is displayed.
Show Variable Size Column `Environment/Watch4Window/Show Size Column` – Boolean	Controls whether the variable size column is displayed.
Show Variable Type Column `Environment/Watch4Window/Show Type Column` – Boolean	Controls whether the variable type column is displayed.

Windows

Property	Description
Buffer Grouping `Environment/Windows/Grouping` – Enumeration	How the files are grouped or listed in the Windows window.

Show File Path as Tooltip `Environment/Windows/Show Filename Tooltips` – Boolean	Show the full file name as a tooltip when hovering over files in the Windows window.
Show Line Count and File Size `Environment/Windows/Show Sizes` – Boolean	Show the number of lines and size of each file in the windows list.

Code Options

Property	Description
`build_output_file_name` – FileName	

Assembler

Property	Description
Additional Assembler Options `asm_additional_options` – StringList	Enables additional options to be supplied to the assembler. This property will have macro expansion applied to it.
Additional Assembler Options From File `asm_additional_options_from_file` – ProjFileNam	Enables additional options to be supplied to the assembler from a file. This property will have macro expansion applied to it.
Assembler `arm_assembler_variant` – Enumeration	Specifies which assembler to use. SEGGER Assembler: Technology preview - For test purposes only.
Backup Additional Assembler Options `asm_additional_options_backup` – String	Value of additional assembler options prior to generic options processing.
Run Preprocessor `arm_preprocess_assembly_code` – Boolean	The assembly code file is preprocessed before assembly

Build

Property	Description
Always Rebuild `build_always_rebuild` – Boolean	Specifies whether or not to always rebuild the project/folder/file.
Batch Build Configurations `batch_build_configurations` – StringList	The set of configurations to batch build.
Build Options Generic File Name `build_generic_options_file_name` – ProjFileName	The file name containing the generic options.
Build Quietly `build_quietly` – Boolean	Suppress the display of startup banners and information messages.
Compilation Database File `compilation_database_file` – FileName	The name of the compilation database file
Delete Intermediate Directory `delete_intermediate_directory` – Boolean	Delete intermediate directory after successful Linking/Archiving.
Dependency File Name `build_dependency_file_name` – FileName	The file name to contain the dependencies.

Enable Unused Symbol Removal `build_remove_unused_symbols` – Enumeration	Enable the removal of unused symbols from the executable.
Exclude From Build `build_exclude_from_build` – Boolean	Specifies whether or not to exclude the project/folder/file from the build.
GCC Prefix `gcc_prefix` – String	The string that is prepended to the gcc toolname e.g arm-none-eabi-. The macro $(GCCPrefix) is set to this value for external build command lines.
GCC Target `gcc_target` – String	The macro $(GCCTarget) is set to this value for build command lines. If clang is used then this value is supplied as the target-triple value.
GCC Version `gcc_version` – String	The macro $(GCCVersion) is set to this value for build command lines.
Generate Compilation Database `generate_compilation_database_file` – Enumerat	Generate a JSON formatted file named by the Compilation Database property.
Generate Dependency File `build_generate_dependency_file` – Enumeration	Generate a dependency file
Include Debug Information `build_debug_information` – Boolean	Specifies whether symbolic debug information is generated.
Inputs File `inputs_file` – FileName	Specifies the inputs file to be used for Linking/Archiving. The files listed in this file will be used rather than the outputs of the project.
Inputs Order `inputs_order` – StringList	Specifies the order to supply input files to be used for Linking/Archiving.
Intermediate Directory `build_intermediate_directory` – DirPath	Specifies a relative path to the intermediate file directory. This property will have macro expansion applied to it. The macro $(IntDir) is set to this value.
Is C++ Project `is_cpp_project` – Enumeration	Supply C++ include directories and libraries to the project build.
Object File Name `build_object_file_name` – FileName	Specifies a name to override the default object file name.
Output Directory `build_output_directory` – DirPath	Specifies a relative path to the output file directory. This property will have macro expansion applied to it. The macro $(OutDir) is set to this value. The macro $(RootRelativeOutDir) is set relative to the **Root Output Directory** if specified.
Preprocess Output File Name `build_preprocess_output_file_name` – FileName	Specifies a name to override the default preprocess output file name.
Project Can Build In Parallel `project_can_build_in_parallel` – Enumeration	Specifies that dependent projects can be built in parallel. Default is No for Staging and Combining project types, Yes for all other project types.

Project Dependencies `project_dependencies` – StringList	Specifies the projects that are built before the current project is built. For Staging project types the dependent projects are built in sequence.
Project Directory `project_directory` – String	Path of the project directory relative to the directory containing the project file. The macro $(ProjectDir) is set to the absolute path of this property.
Project Macros `macros` – StringList	Specifies macro values which are expanded in project properties and for file names in Common configuration only. Each macro is defined as **name=value** and are seperated by **;**.
Project Type `project_type` – Enumeration	Specifies the type of project to build. The options are **Executable, Library, Object file, Staging, Combining, Externally Built Executable, Externally Built Library, Externally Built Object file, Preprocess**.
Property Groups File `property_groups_file_path` – ProjFileName	The file containing the property groups for this project. This is applicable to **Executable** and **Externally Built Executable** project types only.
Root Output Directory `build_root_output_directory` – DirPath	Allows a common root output directory to be specified that can be referenced using the $(RootOutDir) macro.
Suppress Warnings `build_suppress_warnings` – Boolean	Don't report warnings.
Toolchain Directory `build_toolchain_directory` – DirPath	Specify the root of the toolchain directory. This property will have macro expansion applied to it. The macro $(ToolChainDir) is set to this value.
Treat Warnings as Errors `build_treat_warnings_as_errors` – Boolean	Treat all warnings as errors.
Use Clang Not GCC `use_clang_not_gcc` – Boolean	The build will issue clang commands.
Use External GCC `use_external_gcc` – Boolean	The build will issue gcc commands.

Code Analyzer

Property	**Description**
Analyze After Compile `analyze_after_compile` – Boolean	Run the static code analyzer after compile

Analyze Command `analyze_command` – CommandLine	The command to execute for the Analyze action. This property will have macro expansion applied to it with the additional macros: • $(DEFINES) contains a space seperated list of preprocessor definitions as set in the **Preprocessor Definitions** property. • $(INCLUDES) contains a space seperated list of user include directories as set in the **User Include Directories** property.
Analyze Command Options C `analyze_command_c_options` – StringList	Options to supply to the analyze command for C source files.
Analyze Command Options C++ `analyze_command_cpp_options` – StringList	Options to supply to the analyze command for C++ source files.
Clang Tidy Checks C `clang_tidy_checks_c` – StringList	Checks to supply to **clang-tidy** for C source files.
Clang Tidy Checks C++ `clang_tidy_checks_cpp` – StringList	Checks to supply to **clang-tidy** for C++ source files.

Code Generation

Property	Description
ARM ABI Type `arm_abi` – Enumeration	Specifies the ABI type to generate code for. The options are: • **AAPCS** generate calls to the C library to implement floating point operations. • **None** will not specify the ABI.
ARM Advanced SIMD Auto Vectorize `arm_advanced_SIMD_auto_vectorize` – Boolean	Enable automatic code generation for Advanced SIMD.
ARM Advanced SIMD Type `arm_advanced_SIMD_type` – Enumeration	Specifies the Advanced SIMD type to generate code for. The options are: • NEON - Cortex-A based processors

ARM Architecture `arm_architecture` – Enumeration	Specifies the version of the instruction set to generate code for. The options are: • v4T - ARM7TDMI and ARM920T processors • v5TE - ARM9E, Feroceon and XScale processors • v6 - ARM11 processors • v6M - Cortex-M0/M1 processors • v7M - Cortex-M3 processors • v7EM - Cortex-M4/M7 processors • v7R - Cortex-R4/R5/R8 processors • v7A - Cortex-A5/A7/A8/A9/A17 processors • v8R - Cortex-R52 processors • v8A - Cortex-A32/A35/A53/A55/A57/A72/A73/A75 processors • v8M_Baseline - Cortex M23 processor • v8M_Mainline - Cortex M33 processor • v8.1M_Mainline - Cortex-M55/M85 processors • None The corresponding preprocessor definitions: • __ARM_ARCH_4T__ • __ARM_ARCH_5TE__ • __ARM_ARCH_6__ • __ARM_ARCH_6M__ • __ARM_ARCH_7M__ • __ARM_ARCH_7EM__ • __ARM_ARCH_7R__ • __ARM_ARCH_7A__ • __ARM_ARCH_8R__ • __ARM_ARCH_8A__ • __ARM_ARCH_8M_BASELINE__ • __ARM_ARCH_8M_MAINLINE__ • __ARM_ARCH_81M_MAINLINE__ are defined.

ARM Core Type `arm_core_type` – Enumeration	Specifies the core to generate code for. The options are: • ARM7TDMI, ARM7TDMI-S, ARM720T • ARM920T, ARM946E-S, ARM966E-S, ARM968E-S, ARM926EJ-S • ARM1136J-S, ARM1136JF-S, ARM1176JZ-S, ARM1176JZF-S • Cortex-M0, Cortex-M0+, Cortex-M1, Cortex-M23, Cortex-M3, Cortex-M33, Cortex-M4, Cortex-M55, Cortex-M7, Cortex-M85 • Cortex-R4, Cortex-R4F, Cortex-R5, Cortex-R7, Cortex-R8 • Cortex-R52 • Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9, Cortex-A15, Cortex-A17 • Cortex-A32, Cortex-A35, Cortex-A53, Cortex-A55, Cortex-A57, Cortex-A72, Cortex-A73, Cortex-A75 • XScale • None If this property is set to None then the architecture property is used
ARM FP ABI Type `arm_fp_abi` – Enumeration	Specifies the FP ABI type to generate code for. The options are: • **Soft** generate calls to the C library to implement floating point operations. • **SoftFP** generate VFP code to implement floating point operations. • **Hard** generate VFP code to implement floating point operations and use VFP registers to pass floating point parameters on function calls. • **None** will not specify the FP ABI or the FPU.

ARM FPU Type `arm_fpu_type` – Enumeration	Specifies the FPU type to generate code for. The options are: • VFP - ARM9/ARM11 based processors • VFP9 - the same as VFP • VFPv3-D32 - Cortex-A/Cortex-R based processors • VFPv3-D16 - Cortex-A/Cortex-R based processors • VFPv4-D32 - Cortex-A/Cortex-R based processors • VFPv4-D16 - Cortex-A/Cortex-R based processors • FPv4-SP-D16 - Cortex-M4 processors • FPv5-SP-D16 - Cortex-M7/M33/R52 processors • FPv5-D16 - Cortex-M7/M55 processors • FP-ARMv8 - Cortex-A/Cortex-R processors The corresponding preprocessor definitions: • __ARM_ARCH_VFP__ • __ARM_ARCH_VFP3_D32__ • __ARM_ARCH_VFP3_D16__ • __ARM_ARCH_VFP4_D32__ • __ARM_ARCH_VFP4_D16__ • __ARM_ARCH_FPV4_SP_D16__ • __ARM_ARCH_FPV5_SP_D16__ • __ARM_ARCH_FPV5_D16__ • __ARM_ARCH_FP_ARMv8__ are defined.
Additional C++ Modules `gcc_additional_modules` – StringList	Add additional C++ Modules to the module mapper file of the form name=filename.
Byte Order `arm_endian` – Enumeration	Specify the byte order of the target processor. The options are: • **Little** little endian code and data. • **Big** big endian code and data. • **BE-8** little endian code and big endian data. • **None** do not specify the endian.
CM0/CM0+/CM1 Has Small Multiplier `arm_cm0_has_small_multiplier` – Boolean	The CM0/CM0+/CM1 core has the small multiplier.
Code Model (ARM64) `arm64_code_model` – Enumeration	Specify the code model to generate code for.
Data Model (ARM64) `arm64_abi` – Enumeration	Specify the data model to generate code for.
Data Sections `gcc_data_sections` – Boolean	Place each data item in it's own section.

Debugging Level `gcc_debugging_level` – Enumeration	Specifies the level of debugging information to generate. The options are: • None - no debugging information • Level 1 - backtrace and line number debugging information • Level 2 - Level 1 and variable display debugging information • Level 3 - Level 2 and macro display debugging information
Disable Function Inlining `gcc_disable_function_inlining` – Boolean	Disable auto inlining of functions when optimization enables this.
Dwarf Version `gcc_dwarf_version` – Enumeration	Specifies the version of Dwarf debugging information to generate.
Enable Coroutine Support `gcc_enable_coroutines` – Boolean	Specifies whether coroutine support is enabled for C++ programs.
Enable Exception Support `cpp_enable_exceptions` – Enumeration	Specifies whether exception support is enabled for C++ programs.
Enable Linker Relaxation `rv_relax` – Enumeration	If enabled the linker may be able to optimize the instructions used to access symbols.
Enable Modules Support `gcc_enable_modules` – Boolean	Specifies whether modules support is enabled for C++ programs.
Enable Precompiled Header File `gcc_enable_precompiled_header` – Boolean	Enable use of a precompiled header file for the project.
Enable RTTI Support `cpp_enable_rtti` – Enumeration	Specifies whether RTTI support is enabled for C++ programs.
Enable Save Restore `rv_save_restore` – Enumeration	If enabled use smaller but slower prolog and epilog code.
Enable Stack Overflow Prevention `stack_overflow_prevention` – Boolean	Enable Stack Overflow Prevention. For more information read: **https://kb.segger.com/Stack_Overflow_Prevention**
Enable Use Of __cxa_atexit `gcc_use_cxa_at_exit` – Boolean	Enable compiler usage of __cxa_atexit.
Enable Use Of __cxa_guard_acquire/release `gcc_thread_safe_statics` – Boolean	Enable compiler usage of __cxa_guard/acquire/release to enable thread-safe initialization of local statics.
Enumeration Size `gcc_short_enum` – Enumeration	Select between minimal container sized enumerations and int sized enumerations.
FP16 Format. `arm_fp16_format` – Enumeration	The format of 16-bit floating point numbers.
Fast Math `gcc_fast_math` – Enumeration	Enable fast-math mode.

Function Sections `gcc_function_sections` – Boolean	Place each function in it's own section.
GNU Tools Prefix `rv_toolchain_prefix` – String	The RISC-V GNU tools prefix e.g andes-, the macro $(GCCTarget) has this value prepended to it.
Generate Dwarf Debug Address Ranges `gcc_dwarf_generate_aranges` – Boolean	Generate Dwarf .debug_aranges section.
Generate Dwarf Debug Types `gcc_dwarf_generate_debug_types` – Boolean	Generate Dwarf .debug_types section.
Generate Dwarf Pubnames `gcc_dwarf_generate_pubnames` – Boolean	Generate Dwarf .debug_pubnames and .debug_pubtypes sections.
Generate Listing File `asm_generate_listing_file` – Boolean	An source/assembler listing file is generated which can be found in the output files folder
Instruction Set `arm_instruction_set` – Enumeration	Specifies the instruction set to generate code for.
Instrument Functions `arm_instrument_functions` – Boolean	Specifies whether instrumentation calls are generated for function entry and exit.
Is C++ Module `is_cpp_module` – Enumeration	The file contains an importable C++ module unit.
Keep Link Time Optimization Intermediate Files `link_keep_lto_files` – Boolean	Specifies whether to keep the link time optimization resolution and object files.
Link Time Optimization `link_time_optimization` – Boolean	Specifies whether the project should be built for optimization at link time.
Link Time Optimization Additional Options `lto_additional_options` – StringList	Enables additional options to be supplied to the link time optimization process
Long Calls `arm_long_calls` – Boolean	Specifies whether function calls are made using absolute addresses.
Machine Outliner [segger-cc] `clang_machine_outliner` – Enumeration	Select machine outliner mode. An optimization that reduces code size by identifying identical code sequences across functions and replaces them with a call to a function which contains the identical code sequence.
Math Errno `arm_math_errno` – Enumeration	Set errno after calling math functions that are executed with a single instruction, e.g., sqrt.
Merge Globals [segger-cc] `clang_merge_globals` – Boolean	Select whether global declarations are merged. This may reduce code size and increase execution speed for some applications. However, if functions are not used in an application and are eliminated by the linker, merged globals may increase the data size requirement of an application.
No COMMON `gcc_no_common` – Enumeration	Don't put globals in the common section

Omit Frame Pointer `gcc_omit_frame_pointer` – Enumeration	Specifies whether a frame pointer register is omitted if not required.
Optimization Level `gcc_optimization_level` – Enumeration	Specifies the optimization level to use. The options are: • None - don't specify an optimization level • Debug - optimize for debug experience. • Level 0 - no optimization, fastest compilation and best debug experience. • Level 1 - optimize minimally. • Level 2 for speed • Level 2 balanced • Level 2 for size • Level 3 for more speed - optimize even more, will take longer to compile and may produce much larger code.
Outline Atomics (ARM64) `arm64_outline_atomics` – Enumeration	Enable or disable calls to out-of-line helpers to implement atomic operations
Precompiled Header File `gcc_precompiled_header` – Boolean	The precompiled header file for the project.
RISC-V ABI `rv_abi` – Enumeration	Specifies the floating point instruction set to generate code for. The options are: • Default - select abi based on ISA • ilp32 - software floating point • ilp32d - hardware double precision floating point • ilp32e - embedded abi • ilp32f - hardware single precision floating point • lp64 - software floating point • lp64d - hardware double precision floating point • lp64f - hardware single precision floating point • None - do not supply an abi command line option

RISC-V ISA `rv_architecture` – Enumeration	Specifies the instruction set to generate code for. The options are: • rv32e - RV32E with no extensions • rv32ec - RV32E with C extension • rv32ea - RV32E with A extension • rv32eac - RV32E with A and C extension • rv32em - RV32E with M extension • rv32emc - RV32E with M and C extensions • rv32ema - RV32E with M and A extensions • rv32emac - RV32E with M, A and C extensions • rv32i - RV32I with no extensions • rv32ia - RV32I with A extension • rv32ic - RV32I with C extension • rv32iac - RV32I with A and C extensions • rv32im - RV32I with M extension • rv32imc - RV32I with M and C extensions • rv32imf - RV32I with M and F extensions • rv32imfc - RV32I with M, F and C extensions • rv32ima - RV32I with M and A extensions • rv32imac - RV32I with M, A and C extensions • rv32imaf - RV32I with M, A and F extensions • rv32imafc - RV32I with M, A, F and C extensions • rv32g - RV32I with M, A, F and D extensions • rv32gc - RV32I with M, A, F, D and C extensions • rv64i - RV64I with no extensions • rv64ia - RV64I with A extension • rv64ic - RV64I with C extension • rv64im - RV64I with M extension • rv64imc - RV64I with M and C extensions • rv64imf - RV64I with M, and F and C extensions • rv64imfc - RV64I with M, F and C extensions • rv64ima - RV64I with M and A extensions • rv64imac - RV64I with M, A and C extensions • rv64imaf - RV64I with M, A and F extensions • rv64imafc - RV64I with M, A, F and C extensions • rv64g - RV64I with M, A, F and D extensions • rv64gc - RV64I with M, A, F, D and C extensions • None - do not supply an architecture command line option
RISC-V ISA Extension `rv_arch_ext` – String	Additional extensions to supply to the tools
RISC-V ISA P Extension `rv_arch_p` – Boolean	Add support for P extension
RISC-V ISA V Extension `rv_arch_v` – Boolean	Add support for V extension

RISC-V ISA Zba Extension `rv_arch_zba` – Boolean	Add support for Zba extension
RISC-V ISA Zbb Extension `rv_arch_zbb` – Boolean	Add support for Zbb extension
RISC-V ISA Zbc Extension `rv_arch_zbc` – Boolean	Add support for Zbc extension
RISC-V ISA Zbkb Extension `rv_arch_zbkb` – Boolean	Add support for Zbkb extension
RISC-V ISA Zbkx Extension `rv_arch_zbkx` – Boolean	Add support for Zbkx extension
RISC-V ISA Zbs Extension `rv_arch_zbs` – Boolean	Add support for Zbs extension
RISC-V ISA Zcb Extension `rv_arch_zcb` – Boolean	Add support for Zcb extension
RISC-V ISA Zcmp Extension `rv_arch_zcmp` – Boolean	Add support for Zcmp extension
RISC-V ISA Zfinx Extension `rv_arch_zfinx` – Boolean	Add support for Zfinx extension
RISC-V ISA Zicond Extension `rv_arch_zicond` – Boolean	Add support for Zicond extension
RISC-V ISA Zicsr Extension `rv_arch_zicsr` – Boolean	Add support for Zicsr extension
RISC-V ISA Zifencei Extension `rv_arch_zifencei` – Boolean	Add support for Zifencei extension
RISC-V ISA Zmmul Extension `rv_arch_zmmul` – Boolean	Add support for Zmmul extension
RISC-V Memory Code Model `rv_mcmodel` – Enumeration	Specifies the 64-bit code model to generate code for. The options are: • medlow - memory map has address ranges of -2Gb to +2Gb • medany - memory map has address ranges of +2Gb • None - do not supply an mcmodel command line option
Relocation Model [segger-cc] `clang_relocation_model` – Enumeration	Select relocation model.
Signed Char `gcc_signed_char` – Enumeration	The char type is considered to be signed char.
Stack Sizes `generate_stack_sizes` – Boolean	Generate stack sizes section

Strict Aliasing `gcc_strict_aliasing` – Enumeration	Strictest aliasing rules are allowed for optimization.
Strict Align (ARM64) `arm64_strict_align` – Enumeration	Enable or disable generation of unaligned memory accesses
Thread Model [segger-cc] `clang_thread_model` – Enumeration	Select thread model.
Thread-Local Storage Model `gcc_tls_model` – Enumeration	Thread-local storage model.
Thread-Local Storage Pointer `arm_tls_model` – Enumeration	Thread-local storage pointer access model.
Unaligned Access Support `arm_unaligned_access` – Enumeration	Unaligned word and half-words can be accessed. The options are: • **Yes** enable unaligned word and half-words. • **No** disable unaligned word and half-words. • **Auto** disable unaligned word and half-word access for v4T/v5TE/v6M/v8M_Baseline architectures, enable for others.
Unaligned Access Support. `rv_unaligned_access` – Enumeration	Unaligned word and half-words can be accessed. The options are: • **Yes** enable unaligned word and half-words. • **No** disable unaligned word and half-words.
Unwind Tables `arm_unwind_tables` – Enumeration	Generate unwind tables for C code.
Use Builtins `arm_use_builtins` – Enumeration	Use built-in library functions e.g. scanf.
Vector Extension `arm_v81M_mve_type` – Enumeration	Specifies the vector extension type to generate code for. The options are: • MVE - integer instructions • MVE.FP - integer and single precision floating-point instructions
Wide Character Size `gcc_wchar_size` – Enumeration	Select between standard 32-bit or shorter 16-bit size for wide characters and wchar_t.
v7A/v7R Has Integer Divide Instructions `arm_v7_has_divide_instructions` – Boolean	The v7A architecture has integer divide instructions in both ARM and Thumb instruction sets. The v7R architecture has integer divide instructions in the ARM instruction set. The v7R architecture always has integer divide instructions in the Thumb instruction set.
v8.1M Has PACBTI Instructions `arm_v81M_has_pacbti` – Boolean	The v8.1M architecture has PACBTI instructions.
v8A Architecture Extension `arm_v8A_architecture_extension` – Enumeration	The v8A architecture extension.

v8A Has BF16 Instructions `arm_v8A_has_bf16` – Boolean	The v8A architecture has brain half-precision floating point instructions.
v8A Has CRC Instructions `arm_v8A_has_crc` – Boolean	The v8A architecture has CRC instructions.
v8A Has Crypto Instructions `arm_v8A_has_crypto` – Boolean	The v8A architecture has crypto instructions.
v8A Has FP16 Instructions `arm_v8A_has_fp16` – Boolean	The v8A architecture has half-precision floating point instructions.
v8A Has FP16FML Instructions `arm_v8A_has_fp16fml` – Boolean	The v8A architecture has half-precision floating point fmla instructions.
v8A Has I8MM Instructions `arm_v8A_has_i8mm` – Boolean	The v8A architecture has 8-bit integer maxtrix multiply instructions.
v8M Has CMSE Instructions `arm_v8M_has_cmse` – Boolean	The v8M architecture has CMSE instructions.
v8M Has DSP Instructions `arm_v8M_has_dsp` – Boolean	The v8M architecture has DSP instructions.

Combining

Property	Description
Combine Command `combine_command` – CommandLine	The command to execute. This property will have macro expansion applied to it with the macro **$(CombiningOutputFilePath)** set to the output filepath of the combine command and the macro **$(CombiningRelInputPaths)** is set to the (project relative) names of all of the files in the project.
Combine Command Working Directory `combine_command_wd` – String	The working directory in which the combine command is run. This property will have macro expansion applied to it.
Output File Path `combine_output_filepath` – String	The output file path the stage command will create. This property will have macro expansion applied to it.
Set To Read-only `combine_set_readonly` – Enumeration	Set the output file to read only or read/write.

Compiler

Property	Description
Additional C Compiler Only Options `c_only_additional_options` – StringList	Enables additional options to be supplied to the C compiler only. This property will have macro expansion applied to it.

Additional C Compiler Only Options From File `c_only_additional_options_from_file` – ProjFileI	Enables additional options to be supplied to the C compiler only from a file. This property will have macro expansion applied to it.
Additional C++ Compiler Only Options `cpp_only_additional_options` – StringList	Enables additional options to be supplied to the C++ compiler only. This property will have macro expansion applied to it.
Additional C++ Compiler Only Options From File `cpp_only_additional_options_from_file` – ProjF	Enables additional options to be supplied to the C++ compiler only from a file. This property will have macro expansion applied to it.
Additional C/C++ Assembler Options `c_asm_additional_options` – StringList	Enables additional options to be supplied to the assembler when used by the C/C++ compiler. This property will have macro expansion applied to it.
Additional C/C++ Compiler Options `c_additional_options` – StringList	Enables additional options to be supplied to the C/C++ compiler. This property will have macro expansion applied to it.
Additional C/C++ Compiler Options From File `c_additional_options_from_file` – ProjFileName	Enables additional options to be supplied to the C/C++ compiler from a file. This property will have macro expansion applied to it.
Backup Additional C Compiler Only Options `c_only_additional_options_backup` – String	Value of additional C compiler options prior to generic options processing
Backup Additional C++ Compiler Only Options `cpp_only_additional_options_backup` – String	Value of additional C++ compiler options prior to generic options processing
Backup Additional Compiler Options `c_additional_options_backup` – String	Value of additional compiler options prior to generic options processing
C Language Standard `gcc_c_language_standard` – Enumeration	Specifies the language standard to use when compiling C files. The options are: • None - don't specify a language standard • c89/gnu89 • c90/gnu90 • c99/gnu99 • c11/gnu11 • c17/gnu17
C++ Language Standard `gcc_cplusplus_language_standard` – Enumeration	Specifies the language standard to use when compiling C files. The options are: • None - don't specify a language standard • c++98/gnu++98 • c++11/gnu++11 • c++14/gnu++14 • c++17/gnu++17 • c++20/gnu++20 • c++23/gnu++23
Color Diagnostics `compiler_color_diagnostics` – Enumeration	Specifies whether to enable color diagnostic output.

Compile C Files As C++ `c_files_are_cpp` – Boolean	Compile files that have the .c extension with the C++ compiler.
Compiler `arm_compiler_variant` – Enumeration	Specifies which compiler to use.
Compiler Has -Oz `gcc_has_Oz_optimization_level` – Boolean	The compiler support the -Oz optization level.
Enable All Warnings `gcc_enable_all_warnings` – Boolean	Enables the compiler warnings that are specified in the Enable All Warnings (C Compiler Only) Command Line Options properties.
Enable All Warnings C Compiler Only Command Line Options `gcc_c_only_all_warnings_command_line_optio`	The command line options supplied to the C compiler when Enable All Warnings is enabled.
Enable All Warnings C++ Compiler Only Command Line Options `gcc_cpp_only_all_warnings_command_line_opt`	The command line options supplied to the C++ compiler when Enable All Warnings is enabled.
Enable All Warnings Command Line Options `gcc_all_warnings_command_line_options` – Strin	The command line options supplied to the compiler when Enable All Warnings is enabled.
Enforce ANSI Checking `c_enforce_ansi_checking` – Boolean	Perform additional checks for ensure strict conformance to the selected ISO (ANSI) C or C++ standard.
Enforce ANSI Checking C Command Line Options `gcc_c_only_enforce_ansi_checking_command_l`	The command line options supplied to the C compiler when Enforce ANSI Checking is enabled.
Enforce ANSI Checking C++ Command Line Options `gcc_cpp_only_enforce_ansi_checking_command`	The command line options supplied to the C++ compiler when Enforce ANSI Checking is enabled.
Enforce ANSI Checking Command Line Options `gcc_enforce_ansi_checking_command_line_opt`	The command line options supplied to the compiler when Enforce ANSI Checking is enabled.
GNU Version [segger-cc] `clang_gnu_version` – Enumeration	Specifies value of __GNUC__ and related macros. Use the 'Auto' setting to set automatically set this to the version of GCC included in the distribution.
Keep Assembly Source `arm_keep_assembly` – Boolean	Specifies whether assembly code generated by the compiler is kept.
Keep Preprocessor Output `arm_keep_preprocessor_output` – Boolean	Specifies whether preprocessor output generated by the compiler is kept.
No Unsupported Architecture [segger-cc] `segger_cc_no_unsupported_architecture` – Bool	Disable unsupported architecture warning.
Show Caret `compiler_diagnostics_show_caret` – Enumeration	Specifies whether caret is displayed in compiler diagnostics.
Show Messages On Single Lines `compiler_set_message_length_to_zero` – Enumer	Specifies whether messages are shown on single lines.
Supply Absolute File Path `arm_supply_absolute_file_path` – Boolean	Specifies whether absolute file paths are supplied to the compiler.

Supply Execution Character Set `compiler_supply_editor_execute_charset` – Boo	Specifies whether to supply the editor file encoding as the execution character set.
Supply Input Character Set `compiler_supply_editor_input_charset` – Boolea	Specifies whether to supply the editor file encoding as the input character set.
Use Compiler Driver `use_compiler_driver` – Boolean	The build will issue cc commands.

Configuration

Property	Description
Hidden `hidden` – Boolean	Configuration is public.

External Build

Property	Description
Archive Command `external_archive_command` – CommandLine	The command line to archive object files. This property will have macro expansion applied to it with the additional macros: • $(TargetPath) contains the full file name of the **Library File Name** property • $(RelTargePath) contains the project directory relative file name of the **Object File Name** property. • $(Objects) a space seperated list of files to archive, generated from the source files of the project OR. • $(ObjectsFilePath) contains the full file name of the file containing the list of files to archive • $(RelObjectsFilePath) contains the project directory relative file name of the file containing the list of files to link

Assemble Command `external_assemble_command` – CommandLine	The command line to assemble an assembly source file. This property will have macro expansion applied to it with the additional macros: • $(TargetPath) contains the full file name of the **Object File Name** property. • $(RelTargePath) contains the project directory relative file name of the **Object File Name** property. • $(AsmOptions) contains a space seperated list of options as set in the **Additional Assembler Options** property. • $(DependencyPath) contains the filename of the .d file that is required to be output by the compilation for dependency support. • $(RelDependencyPath) contains the relative filename of the .d file that is required to be output by the compilation for dependency support. • $(Defines) contains a space seperated list of preprocessor definitions as set in the **Preprocessor Definitions** property. • $(Undefines) contains a space seperated list of preprocessor undefinitions as set in the **Preprocessor Definitions** property. • $(Includes) contains a space seperated list of user include directories as set in the **User Include Directories** property. • $(IncludeFiles) contains a space seperated list of include files as set in the **Include Files** property.
Build Command `external_build_command` – CommandLine	The command line to build the executable e.g. make. This property will have macro expansion applied to it.

C Compile Command `external_c_compile_command` – CommandLine	The command line to compile a C source file. This property will have macro expansion applied to it with the additional macros: • $(TargetPath) contains the full file name of the **Object File Name** property. • $(RelTargePath) contains the project directory relative file name of the **Object File Name** property. • $(COptions) contains a space seperated list of options as set in the **C Additional C/C++ Compiler Options** property. • $(COnlyOptions) contains a space seperated list of options as set in the **C Additional C Compiler Only Options** property. • $(DependencyPath) contains the filename of the .d file that is required to be output by the compilation for dependency support. • $(RelDependencyPath) contains the relative filename of the .d file that is required to be output by the compilation for dependency support. • $(Defines) contains a space seperated list of preprocessor definitions as set in the **Preprocessor Definitions** property. • $(Undefines) contains a space seperated list of preprocessor undefinitions as set in the **Preprocessor Definitions** property. • $(Includes) contains a space seperated list of user include directories as set in the **User Include Directories** property. • $(IncludeFiles) contains a space seperated list of include files as set in the **Include Files** property.

C++ Compile Command `external_cpp_compile_command` – CommandLine	The command line to compile a C++ source file. This property will have macro expansion applied to it with the additional macros: • $(TargetPath) contains the full file name of the **Object File Name** property. • $(RelTargePath) contains the project directory relative file name of the **Object File Name** property. • $(COptions) contains a space seperated list of options as set in the **C Additional C/C++ Compiler Options** property. • $(CppOnlyOptions) contains a space seperated list of options as set in the **C Additional C++ Compiler Only Options** property. • $(DependencyPath) contains the filename of the .d file that is required to be output by the compilation for dependency support. • $(RelDependencyPath) contains the relative filename of the .d file that is required to be output by the compilation for dependency support. • $(Defines) contains a space seperated list of preprocessor definitions as set in the **Preprocessor Definitions** property • $(Undefines) contains a space seperated list of preprocessor undefinitions as set in the **Preprocessor Definitions** property. • $(Includes) contains a space seperated list of user include directories as set in the **User Include Directories** property. • $(IncludeFiles) contains a space seperated list of include files as set in the **Include Files** property.

C++ Link Command `external_cpp_link_command` – CommandLine	The command line to link an executable. This property will have macro expansion applied to it with the additional macros: • $(TargetPath) contains the full file name of the **Executable File Name** property. • $(RelTargePath) contains the project directory relative file name of the **Executable File Name** property. • $(LinkOptions) contains a space seperated list of options as set in the **Additional Linker Options** property. • $(Objects) a space seperated list of files to link, generated from the source files of the project and the outputs of any dependent projects OR. • $(ObjectsFilePath) contains the full file name of the file containing the list of files to link • $(RelObjectsFilePath) contains the project directory relative file name of the file containing the list of files to link • $(LinkerScriptPath) contains the full file name of the **Linker Script File** property. • $(RelLinkerScriptPath) contains the project directory relative file name of the **Linker Script File** property. • $(MapPath) contains the full file name of the required map file. • $(RelMapPath) contains the project directory relative file name of the required map file.
Clean Command `external_clean_command` – CommandLine	The command line to clean the executable e.g. make clean. This property will have macro expansion applied to it.

Link Command `external_link_command` – CommandLine	The command line to link an executable. This property will have macro expansion applied to it with the additional macros: • $(TargetPath) contains the full file name of the **Executable File Name** property. • $(RelTargePath) contains the project directory relative file name of the **Executable File Name** property. • $(LinkOptions) contains a space seperated list of options as set in the **Additional Linker Options** property. • $(Objects) a space seperated list of files to link, generated from the source files of the project and the outputs of any dependent projects OR. • $(ObjectsFilePath) contains the full file name of the file containing the list of files to link • $(RelObjectsFilePath) contains the project directory relative file name of the file containing the list of files to link • $(LinkerScriptPath) contains the full file name of the **Linker Script File** property. • $(RelLinkerScriptPath) contains the project directory relative file name of the **Linker Script File** property. • $(MapPath) contains the full file name of the required map file. • $(RelMapPath) contains the project directory relative file name of the required map file.
Objects File `external_objects_file_name` – CommandLine	The name of the file containing the list of files to archive or link, generated from the source files of the project.This property will have macro expansion applied to it. The macro $(ObjectsFilePath) is set to this value.

File

Property	Description
File Encoding `file_codec` – Enumeration	Specifies the encoding to use when reading and writing the file.

File Name `file_name` – String	The name of the file. This property will have global macro expansion applied to it. The following macros are set based on the value: $(InputDir) relative directory of file, $(InputName) file name without directory or extension, $(InputFileName) file name, $(InputExt) file name extension, $(InputPath) absolute path to the file name, $(RelInputPath) relative path from project directory to the file name.
File Open Action `file_open_with` – Enumeration	Specifies how to open the file when it is double clicked.
File Type `file_type` – Enumeration	The type of file. Default setting uses the file extension to determine file type.
Flag `file_flag` – Enumeration	Flag which you can use to draw attention to important files in your project.

Folder

Property	Description
Folder Directory `path` – DirPath	Dynamic folder directory specification - ; seperated directory names that will have global macro expansion applied to them.
Folder Exclude `exclude` – StringList	Dynamic folder exclude specification - ; seperated wildcards.
Folder Filter `filter` – String	
Folder Recurse `recurse` – Boolean	Dynamic folder recurse into subdirectories.
Unity Build Exclude Filter `unity_build_exclude_filter` – String	The filter specification to exclude from the unity build - ; seperated wildcards.
Unity Build File Name `unity_build_file_name` – FileName	The file name created that #includes all files in the folder for the unity build.

General

Property	Description
Environment Variables `environment_variables` – StringList	Environment variables to set on solution load.
Inherited Configurations `inherited_configurations` – StringList	The list of configurations that are inherited by this configuration.

Library

Property	Description
DSO Handle Value `link_dso_handle_value` – String	The value to set the __dso_handle symbol to, an empty value will not set the symbol
Debug I/O Implementation `arm_link_debugio_type` – Enumeration	Specifies which Debug I/O mechanism to use for I/O operations. Options are: • **Breakpoint:** Hardware breakpoint instruction and memory locations are used • **DCC:** ARM debug communication channel is used • **Memory Poll:** Memory locations are polled
Enable Locales `LIBRARY_LOCALES` – Enumeration	Enable localization for input/output functions and character classification and conversion. Options are: • **Minimal: small implementation for functions such as isupper() and tolower().**. • **Full:** locale support.
Include Standard Libraries `link_include_standard_libraries` – Boolean	Specifies whether the standard libraries should be linked into your application.
Library ARM Architecture `arm_library_architecture` – Enumeration	Specifies the architecture variant of the library to link with. The default uses the ARM Architecture value
Library Extension Suffix `link_libext_suffix` – String	Specifies a suffix to add to the $(LibExt) macro
Library Heap `LIBRARY_HEAP_TYPE` – Enumeration	Specifies how the library heap is implemented. Options are: • **Basic:** Use low-overhead heap allocator. • **Minimal:** Use alloc-only heap allocator. • **Real-Time:** Use real-time heap allocator. • **None:** Do not implement heap allocator. (Use application-supplied allocator).
Library Heap Locking `LIBRARY_HEAP_LOCKING` – Enumeration	Specifies how the library heap locking is implemented. Options are: • **Disable Interrupts:** Disable global interrupts. • **User:** Call user supplied __heap_lock/ __heap_unlock functions.

Library I/O `LIBRARY_IO_TYPE` – Enumeration	Specifies how the library does I/O. Options are: • **RTT:** Use SEGGER Real-Time Transfer for I/O operations without halting the system. Recommended for maximum speed. • **SEMIHOST:** Format output and write to RAM buffer. Halt CPU for I/O operation. Provides hosted file I/O. • **SEMIHOST (host-formatted):** Halt CPU for I/O operation. Recommended for minimum size. • **SWO:** Format output and write to RAM buffer. Use SWO for Output operation. Use CMSIS ITM_RxBuffer/ITM_RXBUFFER_EMPTY memory access for Input Operation. • **SWO (Interrupts Disabled):** Format output and write to RAM buffer. Use SWO for Output operation with interrupts disabled. Use CMSIS ITM_RxBuffer/ITM_RXBUFFER_EMPTY memory access for Input Operation. • **None:** Do not include I/O implementation. Use user-supplied I/O Mechanism. For more information read: **https://kb.segger.com/Embedded_Studio_Library_IO**
Library Instruction Set `arm_library_instruction_set` – Enumeration	Specifies the instruction set variant of the libraries to link with, **Default** will use the **Instruction Set** value.
Library Optimization `arm_library_optimization` – Enumeration	Specifies whether to link with libraries optimized for speed or size.
Library Optimization `rv_library_optimization` – Enumeration	Specifies whether to link with libraries optimized for speed or size.
Standard Libraries Configuration Prefix `link_standard_libraries_configuration_pref`	Specifies the prefix to prepend to the library build configuration.
Standard Libraries Directory `link_standard_libraries_directory` – String	Specifies where to find the standard libraries

Linker

Property	Description
Additional Input Files `linker_additional_files` – StringList	Enables additional object and library files to be supplied to the linker.
Additional Linker Options `linker_additional_options` – StringList	Enables additional options to be supplied to the linker.

Additional Linker Options From File `linker_additional_options_from_file` – ProjFile	Enables additional options to be supplied to the linker from a file.
Additional Linker Script Generator Options `arm_additional_mkld_options` – StringList	Enables additional options to be supplied to the linker script generator.
Additional Output Format `linker_output_format` – Enumeration	The format used when creating an additional linked output file.The options are: • **None** do not create an additional output file. • **bin** create a binary file. • **srec** create a Motorola S-Record file. • **hex** create an Intel Hex file.
Additional System Libraries `linker_additional_system_libraries` – StringList	Enables additional system libraries to be supplied to the linker.
Align ro Section [segger-ld] `link_align_ro` – Enumeration	Specifies the linker byte alignment of the ro section
Align rw Section [segger-ld] `link_align_rw` – Enumeration	Specifies the linker byte alignment of the rw section
Align rx Section [segger-ld] `link_align_rx` – Enumeration	Specifies the linker byte alignment of the rx section
Align zi Section [segger-ld] `link_align_zi` – Enumeration	Specifies the linker byte alignment of the zi section
Allow Multiple Symbol Definition `arm_linker_allow_multiple_definition` – Boolea	Do not report error if the same symbol is defined more than once in object files/libraries.
Backup Additional Linker Options `link_additional_options_backup` – String	Value of additional linker options prior to generic options processing
Breakpad Symbols Directory `linker_breakpad_symbols_directory` – String	Specifies location of the breakpad symbols directory.
CMSE Import Library File `arm_linker_cmse_import_library_file_name` – F	Specifies the name of the CMSE import library to generate.
Check CMSE Import Library File `arm_linker_check_cmse_import_library_file_`	Specifies the name of the file to check the generated CMSE import library with.
Check For Memory Section Overflow `arm_library_check_memory_section_overflow` –	Specifies whether the linker should check whether program sections exceed their specified size.
Check For Memory Segment Overflow `arm_library_check_memory_segment_overflow` –	Specifies whether the linker should check whether program sections fit in their memory segments.
Deduplicate Code Sections [segger-ld] `link_dedupe_code` – Enumeration	Specifies whether the linker finds readonly code sections that are identical and discard duplicates.
Deduplicate Data Sections [segger-ld] `link_dedupe_data` – Enumeration	Specifies whether the linker finds readonly data sections that are identical and discard duplicates.

Default Fill Pattern `arm_linker_script_generator_default_fill_p`	Specifies the default pattern used to fill unspecified regions of memory in a generated linker script. This pattern maybe overidden by the *fill* attribute of a program section in the section placement file.
Emit Relocations `arm_linker_emit_relocations` – Enumeration	Output relocation information into the executable.
Enable Call Springboarding [segger-ld] `rv_spring_board` – Enumeration	If enabled the linker may be able to optimize the instructions used for function calls.
Enable Outline Optimization [segger-ld] `rv_outline` – Enumeration	If enabled the linker may be able to reduce code size by extracting common code into subroutines.
Enable Tail Merging [segger-ld] `rv_tail_merge` – Enumeration	If enabled the linker may be able to reduce code size by extracting common code immediately before function return.
Entry Point `gcc_entry_point` – String	Specifies the entry point of the program. None will not supply an entry point to the linker.
Gap Fill Value [segger-ld] `arm_linker_additional_output_file_gap_fill`	The value to fill unused space between sections. This is supplied to segger-ld and the additional output file creation build step.
Generate Breakpad Symbols `linker_generate_breakpad_symbols` – Boolean	Specifies whether to generate breakpad symbols from the linked image.
Generate Linker Map File `linker_map_file` – Boolean	Specifies whether to generate a linkage map file.
Generate Linker Script `arm_generate_linker_script` – Boolean	Specifies whether a linker script file should be generated from the section placement and/or memory map files.
Generate Log File [segger-ld] `linker_log_file` – Boolean	Specifies whether to generate a linkage log file.
Generate Map File [segger-ld] `link_map_file` – Enumeration	Specifies whether to generate a linkage map file.
Indirect File Name `linker_indirect_file_name` – String	The name of the generated indirect file.
Indirect File Supported `linker_use_indirect_files` – Enumeration	Linker can use @indirect file for input files.
Inline Small Functions [segger-ld] `link_inline` – Enumeration	Specifies whether the linker inlines small functions at the call site rather than calling the function.
Keep Indirect Files `linker_keep_indirect_files` – Boolean	Keep generated linker indirect files.
Keep Symbols `linker_keep_symbols` – StringList	Specifies the symbols that should be kept by the linker even if they are not reachable.
Link Dependent Projects `link_dependent_projects` – Boolean	Specifies whether to link the output of dependent library projects.

Link Whole Archive `arm_linker_whole_archive` – StringList	List the archives that require to be linked in whole.
Linker `arm_linker_variant` – Enumeration	Specifies which linker to use.
Linker Map File Name `linker_map_file_name` – FileName	The file name to contain the linkage map file.
Linker Script File `link_linker_script_file` – ProjFileName	The name of the manual linker script file.
Linker Search Path `arm_linker_search_path` – StringList	Specify the linker script search path.
Linker Symbol Definitions `link_symbol_definitions` – StringList	Specifies one or more linker symbol definitions.
List All Undefineds [segger-ld] `link_list_all_undefineds` – Enumeration	Specifies whether the linker issues one error for each undefined symbol.
Map File Format [segger-ld] `link_map_file_format` – Enumeration	Specifies map file format generated by the linker.
Memory Map File `linker_memory_map_file` – ProjFileName	The name of the file containing the memory map description.
Memory Map Macros `linker_memory_map_macros` – StringList	Macro values to substitue in memory map nodes. Each macro is defined as **name=value** and are seperated by **;**.
Memory Segments `linker_section_placements_segments` – StringList	The start, access and size of named segments in the target, these are used when no memory map file is available.Each segment is specified by NAME RWX HEXSTART HEXSIZE for example FLASH RX 0x08000000 0x00010000
Merge Sections [segger-ld] `link_merge_sections` – Enumeration	Specifies whether the linker merges compatible sections.
Merge String Constants [segger-ld] `link_merge_strings` – Enumeration	Specifies whether the linker merges duplicate string constants.
No Enum Size Warning `arm_linker_no_enum_size_warning` – Boolean	Do not generate warnings when object files have different ARM EABI enum size attributes.
No Start File `arm_linker_no_start_files` – Boolean	Do not use startup files when linking.
No Wide Char Size Warning `arm_linker_no_wchar_size_warning` – Boolean	Do not generate warnings when object files have different ARM EABI wide character size attributes.
Pad ro Section [segger-ld] `link_pad_ro` – Boolean	Specifies whether the linker pads the ro section
Pad rw Section [segger-ld] `link_pad_rw` – Boolean	Specifies whether the linker pads the rw section

Pad rx Section [segger-ld] `link_pad_rx` – Boolean	Specifies whether the linker pads the rx section
Pad zi Section [segger-ld] `link_pad_zi` – Boolean	Specifies whether the linker pads the zi section
Print Memory Usage `linker_print_memory_usage` – Boolean	Print summary of memory used by the application.
Section Headers [segger-ld] `link_section_headers` – Enumeration	Specifies whether the linker produces a section header table.
Section Placement File `linker_section_placement_file` – ProjFileName	The name of the file containing section placement description.
Section Placement Macros `linker_section_placement_macros` – StringList	Macro values to substitue in section placement nodes - MACRO1=value1;MACRO2=value2.
Start/End Group Required `linker_requires_start_group` – Boolean	Linker requires --start-group and --end-group for input files.
Strip Debug Information `linker_strip_debug_information` – Boolean	Specifies whether debug information should be stripped from the linked image.
Strip Symbols `gcc_strip_symbols` – Boolean	Specifies whether symbols should be stripped.
Supply Library Linker Symbols `link_supply_library_linker_symbols` – Boolean	Specifies whether the library symbols are supplied to the linker command line.
Supply Memory Segments To Linker [segger-ld] `linker_supply_memory_segments` – Boolean	Specifies whether to supply memory segments on the linker command line.
Suppress Warning on Executable Stack `arm_linker_no_warn_on_executable_stack` – Boo	No warning on executable stack.
Suppress Warning on Mismatch `arm_linker_no_warn_on_mismatch` – Boolean	No warning on mismatched object files/libraries.
Suppress Warning on RWX Segments `arm_linker_no_warn_on_rwx_segments` – Boolean	No warning on RWX segments.
Symbols File `arm_linker_symbols_files` – FileName	Specify the name of a symbols file to link.
Thread Pointer Model [segger-ld] `rv_thread_pointer_model` – Enumeration	Controls how the thread pointer register is used.
Treat Libraries As Object Files `linker_treat_libraries_as_object_files` – Boo	Specifies whether the linker treats libraries as a set of object files.
Treat Linker Warnings as Errors `arm_linker_treat_warnings_as_errors` – Boolear	Treat linker warnings as errors.
Use Manual Linker Script `link_use_linker_script_file` – Boolean	Specifies whether to use a manual linker script.

Package

Property	Description
Package Dependencies `package_dependencies` – StringList	Specifies the packages the current project depends upon.
Package Directory `package_directory` – DirPath	Specifies the directory packages are installed to. If no directory is specified, the default package directory is used.

Preprocessor

Property	Description
Add Property Group Options `add_property_group_includes_defines` – Boolear	Supply the defines and includes that are selected by the property group.
Ignore Includes `c_ignore_includes` – Boolean	Ignore the include directories properties.
Include Files `c_include_files` – StringList	Specifies the list of files to include before preprocessing. This property will have macro expansion applied to it.
Include Files Assembler Only `c_include_files_asm_only` – StringList	Specifies the list of files to include before preprocessing. This property will have macro expansion applied to it.
Include Files C Compiler Only `c_include_files_c_only` – StringList	Specifies the list of files to include before preprocessing. This property will have macro expansion applied to it.
Include Files C++ Compiler Only `c_include_files_cpp_only` – StringList	Specifies the list of files to include before preprocessing. This property will have macro expansion applied to it.
Macro Files `c_macros_files` – StringList	Specifies the list of macro files to include before preprocessing. This property will have macro expansion applied to it.
Macro Files Assembler Only `c_macros_files_asm_only` – StringList	Specifies the list of macro files to include before preprocessing. This property will have macro expansion applied to it.
Macro Files C Compiler Only `c_macros_files_c_only` – StringList	Specifies the list of macro files to include before preprocessing. This property will have macro expansion applied to it.
Macro Files C++ Compiler Only `c_macros_files_cpp_only` – StringList	Specifies the list of macro files to include before preprocessing. This property will have macro expansion applied to it.

Preprocessor Definitions `c_preprocessor_definitions` – StringList	Specifies one or more preprocessor definitions. This property will have macro expansion applied to it.
Preprocessor Definitions Assembler Only `c_preprocessor_definitions_asm_only` – StringLi	Specifies one or more preprocessor definitions. This property will have macro expansion applied to it.
Preprocessor Definitions C Compiler Only `c_preprocessor_definitions_c_only` – StringList	Specifies one or more preprocessor definitions. This property will have macro expansion applied to it.
Preprocessor Definitions C++ Compiler Only `c_preprocessor_definitions_cpp_only` – StringLi	Specifies one or more preprocessor definitions. This property will have macro expansion applied to it.
Preprocessor Undefinitions `c_preprocessor_undefinitions` – StringList	Specifies one or more preprocessor undefinitions. This property will have macro expansion applied to it.
Preprocessor Undefinitions Assembler Only `c_preprocessor_undefinitions_asm_only` – Strin	Specifies one or more preprocessor undefinitions. This property will have macro expansion applied to it.
Preprocessor Undefinitions C Compiler Only `c_preprocessor_undefinitions_c_only` – StringLi	Specifies one or more preprocessor undefinitions. This property will have macro expansion applied to it.
Preprocessor Undefinitions C++ Compiler Only `c_preprocessor_undefinitions_cpp_only` – Strin	Specifies one or more preprocessor undefinitions. This property will have macro expansion applied to it.
System Include Directories `c_system_include_directories` – StringList	Specifies the system include path. This property will have macro expansion applied to it.
Undefine All Preprocessor Definitions `c_undefine_all_preprocessor_definitions` – Bo	Does not define any standard preprocessor definitions.
User Include Directories `c_user_include_directories` – StringList	Specifies the user include path. This property will have macro expansion applied to it.
User Include Directories Assembler Only `c_user_include_directories_asm_only` – StringLi	Specifies the user include path. This property will have macro expansion applied to it.
User Include Directories C Compiler Only `c_user_include_directories_c_only` – StringList	Specifies the user include path. This property will have macro expansion applied to it.
User Include Directories C++ Compiler Only `c_user_include_directories_cpp_only` – StringLi	Specifies the user include path. This property will have macro expansion applied to it.

Printf/Scanf

Property	Description
Printf Floating Point Supported `linker_printf_fp_enabled` – Enumeration	Are floating point numbers supported by the printf function group.
Printf Integer Support `linker_printf_fmt_level` – Enumeration	The largest integer type supported by the printf function group.
Printf Width/Precision Supported `linker_printf_width_precision_supported` – Bo	Enables support for width and precision specification in the printf function group.
Scanf Classes Supported `linker_scanf_character_group_matching_enab`	Enables support for %[...] and %[^...] character class matching in the scanf functions.

Scanf Floating Point Supported `linker_scanf_fp_enabled` – Boolean	Are floating point numbers supported by the scanf function group.
Scanf Integer Support `linker_scanf_fmt_level` – Enumeration	The largest integer type supported by the scanf function group.
Wide Characters Supported `linker_printf_wchar_enabled` – Boolean	Are wide characters supported by the printf function group.

Project

Property	Description
Flag `project_flag` – Enumeration	Flag which you can use to draw attention to important projects in your solution.

Runtime Memory Area

Property	Description
`arm_linker_stack_size` – IntegerRange	
Heap Size `arm_linker_heap_size` – IntegerRange	The size of the heap in bytes. The size must be a multiple of 8. The preprocessor define __HEAP_SIZE__ is set to this value.
Heap Size `heap_size` – IntegerRange	The size of the heap in bytes. The size must be a multiple of 16.
Process Stack Size `arm_linker_process_stack_size` – IntegerRange	The size of the process stack in bytes. The size must be a multiple of 8.
Stack Size `stack_size` – IntegerRange	The size of the stack in bytes. The size must be a multiple of 16.
Stack Size (Abort Mode) `arm_linker_abt_stack_size` – IntegerRange	The size of the Abort mode stack in bytes. The size must be a multiple of 8.
Stack Size (FIQ Mode) `arm_linker_fiq_stack_size` – IntegerRange	The size of the FIQ mode stack in bytes. The size must be a multiple of 8.
Stack Size (IRQ Mode) `arm_linker_irq_stack_size` – IntegerRange	The size of the IRQ mode stack in bytes. The size must be a multiple of 8.
Stack Size (Supervisor Mode) `arm_linker_svc_stack_size` – IntegerRange	The size of the Supervisor mode stack in bytes. The size must be a multiple of 8.
Stack Size (Undefined Mode) `arm_linker_und_stack_size` – IntegerRange	The size of the Undefined mode stack in bytes. The size must be a multiple of 8.

Section

Property	Description
Code Section Name `default_code_section` – String	Specifies the default name to use for the program code section.
Constant Section Name `default_const_section` – String	Specifies the default name to use for the read-only constant section.
Data Section Name `default_data_section` – String	Specifies the default name to use for the initialized, writable data section.
ISR Section Name `default_isr_section` – String	Specifies the default name to use for the ISR code.
Vector Section Name `default_vector_section` – String	Specifies the default name to use for the interrupt vector section.
Zeroed Section Name `default_zeroed_section` – String	Specifies the default name to use for the zero-initialized, writable data section.

Solution

Property	Description
Flag `solution_flag` – Enumeration	Flag which you can use to draw attention to important projects in your solution.
Properties Filter `properties_filter` – StringList	The names of project properties that can be displayed at the solution

Source Code

Property	Description
Additional Code Completion Compiler Options `code_completion_options` – StringList	Additional source indexing and code completion compiler options.
Inhibit Source Indexing `project_inhibit_indexing` – Boolean	Disable source indexing and code completion for files/folders/projects that would normally be indexed (C/C++ files in executable and library projects).
Source Code Control Directory `source_code_control_directory` – DirPath	Source code control directory root.

Staging

Property	Description

Output File Path `stage_output_filepath` – String	The output file path the stage command will create. This property will have macro expansion applied to it.
Output File Path Replace `stage_output_filepath_replace` – String	A string to search for and optionally a string to replace it with. The search and replace strings are separated with a semicolon. If no replace string is supplied then the search string is removed. This property will have macro expansion applied to it.
Set To Read-only `stage_set_readonly` – Enumeration	Set the output file permissions to read only or read/write.
Stage Command `stage_command` – CommandLine	The command to execute. This property will have macro expansion applied to it with the additional **$(StageOutputFilePath)** macro set to the output filepath of the stage command.
Stage Command Working Directory `stage_command_wd` – String	The working directory in which the stage command is run. This property will have macro expansion applied to it.
Stage Project Command `stage_post_build_command` – CommandLine	The command to execute after staging commands have executed. This property will have macro expansion applied to it.
Stage Project Command Working Directory `stage_post_build_command_wd` – String	The working directory where the post build command runs. This property will have macro expansion applied to it.

User Build Step

Property	Description
Link Patch Command `linker_patch_build_command` – CommandLine	A command to run after the link but prior to additional binary file generation. This property will have macro expansion applied to it with the additional **$(TargetPath)** macro set to the output filepath of the linker command.
Link Patch Working Directory `linker_patch_build_command_wd` – DirPath	The working directory where the link patch command is run. This property will have macro expansion applied to it.
Post-Archive Command `archive_post_build_command` – CommandLine	A command to run after the archive command has completed.This property will have macro expansion applied to it with the additional **$(TargetPath)** macro set to the output filepath of the archive command.
Post-Archive Working Directory `archive_post_build_command_wd` – DirPath	The working directory where the post-archive command is run. This property will have macro expansion applied to it.

Post-Build Command `post_build_command` – CommandLine	The command to execute after a project build. This property will have macro expansion applied to it.
Post-Build Command Control `post_build_command_control` – Enumeration	Controls when the post-build command is run, either **Always Run** or when **Run When Build Has Occurred**.
Post-Build Command Working Directory `post_build_command_wd` – String	The working directory in which the post-build command is run. This property will have macro expansion applied to it.
Post-Compile Command `compile_post_build_command` – CommandLine	A command to run after the compile command has completed. This property will have macro expansion applied to it with the additional **$(TargetPath)** macro set to the output filepath of the compiler command.
Post-Compile Working Directory `compile_post_build_command_wd` – DirPath	The working directory where the post-compile command is run. This property will have macro expansion applied to it.
Post-Link Command `linker_post_build_command` – CommandLine	A command to run after the link command has completed.This property will have macro expansion applied to it with the additional **$(TargetPath)** macro set to the output filepath of the linker command and **$(PostLinkOutputFilePath)** set to the value of the output filepath of the post link command.
Post-Link Output File `linker_post_build_command_output_file` – Strin	The name of the file created by the post-link command. This property will have macro expansion applied to it.
Post-Link Working Directory `linker_post_build_command_wd` – DirPath	The working directory where the post-link command is run. This property will have macro expansion applied to it.
Pre-Build Command `pre_build_command` – CommandLine	The command to execute before a project build. This property will have macro expansion applied to it.
Pre-Build Command Control `pre_build_command_control` – Enumeration	Controls when the pre-build command is run, either **Always Run** or when **Run When Build Required**.
Pre-Build Command Working Directory `pre_build_command_wd` – String	The working directory in which the pre-build command is run. This property will have macro expansion applied to it.
Pre-Compile Command `compile_pre_build_command` – CommandLine	A command to run before the compile command. This property will have macro expansion applied to it.
Pre-Compile Command Output File Path `compile_pre_build_command_output_file_name`	The pre-compile generated file name. This property will have macro expansion applied to it.
Pre-Compile Working Directory `compile_pre_build_command_wd` – DirPath	The working directory where the pre-compile command is run. This property will have macro expansion applied to it.
Pre-Link Command `linker_pre_build_command` – CommandLine	A command to run before the link command. This property will have macro expansion applied to it.

Pre-Link Command Output File Path `linker_pre_build_command_output_file_name` –	The pre-link generated file name. This property will have macro expansion applied to it.
Pre-Link Working Directory `linker_pre_build_command_wd` – DirPath	The working directory where the pre-link command is run. This property will have macro expansion applied to it.

Debug Options

Debugger

Property	Description
Alternative LDR Disassembly `debug_alternative_ldr_dis` – Boolean	Show alternative disassembly of ldr*/vldr instructions
CPU Register File `debug_cpu_registers_file` – ProjFileName	The name of the file containing CPU register definitions.
Command Arguments `debug_command_arguments` – String	The command arguments passed to the executable. This property will have macro expansion applied to it.
Coprocessor Disassembly `debug_custom_cp_dis` – Enumeration	Show disassembly coprocessor instructions
Debug Additional Configurations `debug_additional_configurations` – StringList	The debugger will debug the project in the specified additional configurations.
Debug Additional Projects `debug_dependent_projects` – StringList	The debugger will load (if not already loaded by Load Additional Projects) and debug the specified additional projects.
Debug Project Name `debug_project_name` – String	The name of the project used by the debugger when debugging multiple projects
Debug Symbols File[0] `external_debug_symbols_file_name` – ProjFileNam	The name of the debug symbols file. This property will have macro expansion applied to it. If it is not defined then the main load file is used.
Debug Symbols File[1] `external_debug_symbols_file_name1` – ProjFileNa	The name of the debug symbols file. This property will have macro expansion applied to it. If it is not defined then the main load file is used.
Debug Symbols File[2] `external_debug_symbols_file_name2` – ProjFileNa	The name of the debug symbols file. This property will have macro expansion applied to it. If it is not defined then the main load file is used.
Debug Symbols File[3] `external_debug_symbols_file_name3` – ProjFileNa	The name of the debug symbols file. This property will have macro expansion applied to it. If it is not defined then the main load file is used.
Debug Symbols Load Address[0] `external_debug_symbols_load_address` – String	The (code) address to be added to the debug symbol (code) addresses.
Debug Symbols Load Address[1] `external_debug_symbols_load_address1` – String	The (code) address to be added to the debug symbol (code) addresses.
Debug Symbols Load Address[2] `external_debug_symbols_load_address2` – String	The (code) address to be added to the debug symbol (code) addresses.
Debug Symbols Load Address[3] `external_debug_symbols_load_address3` – String	The (code) address to be added to the debug symbol (code) addresses.

Debug Terminal Log File `debug_terminal_log_file` – Unknown	A file to write the output from the debug terminal to.
Default Debug I/O implementation `arm_debugIO_Implementation` – Enumeration	The default Debug I/O implementation used by the debugger if symbols are unavailable.
Display DCC data `arm_display_DCC` – Boolean	The debugger will display data that is written to the DCC when debugIO is not used.
Entry Point Symbol `debug_entry_point_symbol` – String	Debugger will start execution at symbol if defined.
HTML Watch File `debug_html_watch` – ProjFileName	The file used by the debugger's HTML Watch window, see $(StudioDir)/html/heap.htm for example.
Has Hypervisor Mode `arm_has_hypervisor_mode` – Boolean	Show hypervisor mode registers
Has Monitor Mode `arm_has_monitor_mode` – Boolean	Show monitor mode registers
Has Vector Catch `arm_has_vector_catch` – Boolean	Vector catching is supported
ISA Extensions Debug `rv_debug_extensions` – Enumeration	Specify ISA extensions supported by debugger.
Ignore .debug_aranges Section `debug_ignore_debug_aranges` – Boolean	The debugger will not use the .debug_aranges section.
Ignore .debug_frame Section `debug_ignore_debug_frame` – Boolean	The debugger will not use the .debug_frame section.
Load Additional Projects `debug_load_additional_projects` – StringList	The debugger will load the outputs of the specified additional projects.
Memory Upload Page Size `debug_memory_upload_page_size` – Integer	The aligned page size the debugger uses when uploading address ranges.
NVIC Interrupt Sources `arm_interrupt_names` – StringList	The interrupt names in the NVIC registers for the debugger to display.
NVIC Num Priority Bits `arm_num_interrupt_priority_bits` – Integer	The number of interrupt priority bits in the NVIC registers for the debugger to display
RTT Channel Number `debug_RTT_channel` – IntegerRange	The RTT channel number to service in the debug terminal.
RTT Control Block Address `debug_RTTCB` – String	The symbol or 0x prefixed address of the RTT control block.
RTT Enable `debug_enable_RTT` – Boolean	If enabled the debugger will service RTT input/output in the debug terminal.
Register Definition File `debug_register_definition_file` – ProjFileName	The name of the file containing register definitions.
Register Definition File Type `debug_register_definition_file_type` – Enumer	The type of the file containing register definitions.

Reserved Member Name `reservedMember_name` – String	The struct reserved member name. Struct members that contain the (case insensitive) string will not be displayed.
Restrict Memory Access `debug_restrict_memory_access` – Boolean	If enabled the debugger will only display variables and backtrace in the address ranges of the memory map or the sections in the elf file.
Run To `debug_initial_breakpoint` – String	The initial breakpoint to set
Run To Control `debug_initial_breakpoint_set_option` – Enumer	Specify when the initial breakpoint should be set
Start Address `external_start_address` – String	The address to start the externally built executable running from.
Start From Entry Point Symbol `debug_start_from_entry_point_symbol` – Enumer	If **Yes** the debugger will start execution from the entry point symbol. If **No** the debugger will start execution from the target specific location. If **Don't** the debugger will not start execution.
Starting Stack Pointer Value `debug_stack_pointer_start` – String	The symbol or 0x prefixed value to set the stack pointer on start debugging.
Startup Completion Point `debug_startup_completion_point` – StringList	Specifies the point in the program where startup is complete. Software breakpoints and debugIO will be enabled after this point has been reached.
Target Connection `debug_target_connection` – Enumeration	Specifies the target to connect to for debugging actions.
Target Connection String `debug_target_connection_string` – Enumeration	A target interface specific connection string.
Target Device [ARM] `arm_target_device_name` – String	The name of the device to connect to. This property will have macro expansion applied to it. The macro $(DeviceName) is set to the value of this property.
Target Device [RISC-V] `target_device_name` – String	The name of the device to connect to. This property will have macro expansion applied to it. The macro $(DeviceName) is set to the value of this property.
Thread Maximum `debug_threads_max` – IntegerRange	The maximum number of threads to display.
Threads Script File `debug_threads_script` – ProjFileName	The threads script used by the debugger.
Type Interpretation File `debug_type_file` – FileName	Specifies the type interpretation file to use.
Type Interpretation File 2 `debug_type_file_2` – String	Specifies the second type interpretation file to use.
Working Directory `debug_working_directory` – DirPath	The working directory for a debug session. This property will have macro expansion applied to it.

GDB Server

Property	Description
Allow Memory Access During Execution `gdb_server_allow_memory_access_during_exec`	Specifies whether memory can be access while target is running. If set to *No*, target will be stopped each time memory is accessed.
Auto Start GDB Server `gdb_server_autostart_server` – Boolean	Specifies whether a GDB server should be started on connect.
Breakpoint Types `gdb_breakpoint_types` – Enumeration	Specifies the type of breakpoints to use.
CSR Registers [RISC-V] `gdb_server_csr_registers` – StringList	Specifies mapping from CSR register numbers to GDB server register names, for example "2000=csr_mmisc_ctl".
Connect Timeout `gdb_server_connect_timeout` – Integer	The length of time in seconds to attempt to connect to server before failing.
GDB Server Command Line `gdb_server_command_line` – CommandLine	The command line to start the gdb server
Host `gdb_server_hostname` – String	The hostname of the GDB server to connect to.
Ignore Checksum Errors `gdb_server_ignore_checksum_errors` – Boolean	Specifies whether an incorrect GDB server checksum causes and error.
Log File `gdb_server_log_file` – Unknown	Specifies a file to output a log of GDB server transactions to.
Port `gdb_server_port` – Integer	The port number to use to connect to the GDB server.
Read Timeout `gdb_server_read_timeout` – Integer	The length of time in seconds to attempt to read from server before failing.
Register Access `gdb_server_register_access` – Enumeration	Specifies how registers are accessed
Reset and Stop Command `gdb_server_reset_command` – String	The remote GDB server command to use to reset and stop the target.
Target XML File `gdb_server_target_xml_file` – FileName	If specified, points to a target.xml file to use in place of file returned by the GDB server.
Type `gdb_server_type` – Enumeration	Specifies the type of GDB server being connected to. J-Link, OpenOCD, ST-LINK and pyOCD gdb server implementations are currently supported.
Write Timeout `gdb_server_write_timeout` – Integer	The length of time in seconds to attempt to write to server before failing.

J-Link

Property	Description
Additional J-Link Options `JLinkExecuteCommand` – StringList	Specify additional J-Link options to allow enabling or disabling advanced features and fine tuning. For more information see **J-Link Command Strings**
Enable Adaptive Clocking `adaptive` – Enumeration	Adaptive clocking is enabled.
Erase All `target_loader_erase_all` – Enumeration	If set to **Yes**, all of the FLASH memory on the target will be erased prior to downloading the application. If set to **No**, only the areas of FLASH containing the program being downloaded will be erased. If set to **Default** the behaviour is target specific.
Exclude Flash Cache Range `JLinkExcludeFlashCacheRange` – String	Address range to exclude from flash cache. This can be specified by either 'start_address-end_address' or 'address,size'. For example: 0x08000000,0x10000.
Host Connection `Connection` – Enumeration	Defines how to connect the host to the J-Link: • "USB": Connect to J-Link via USB • "USB S/N": Connect to J-Link with specified serial number via USB e.g. USB 174300001 • "IP S/N": Connect to J-Link with specified serial number via IP e.g. IP 174300001 • "IP n.n.n.n port": Connect to J-Link with specified IP address and optional port number e.g. IP 192.168.20.20 19020
JTAG Instruction Register Size Before Target `arm_linker_jtag_pad_post_ir` – IntegerRange	Specifies the number of bits in the instruction register before the target (as seen from TDI), which is the number of bits to pad the JTAG instruction register with the BYPASS instruction after the target instruction.
JTAG Number Of Devices Before Target `arm_linker_jtag_pad_post_dr` – IntegerRange	Specifies the number of devices before the target (as seen from TDI), which is the number of bits to pad the JTAG data register.
Log File `JLinkLogFileName` – FileName	The file to output the J-Link log to.
Script File `JLinkScriptFileName` – FileName	The file path of the optional J-Link script file to use.
Show Log Messages In Output Window `JLinkShowLog` – Boolean	Display the J-Link log messages to the output window.
Speed `speed` – IntegerRange	The required JTAG clock frequency in kHz (0 to auto-detect best possible).

Supply Power `supplyPower` – Boolean	The J-Link supplies power to the target.
Target Has Cycle Counter [RISC-V] `riscv_has_cycle_counter` – Boolean	The target device implements the mcycle CSR.
Target Interface Type [ARM] `arm_target_interface_type` – Enumeration	Specifies the type of interface the target has. The options are: • JTAG - Use JTAG interface • SWD - Use SWD interface • cJTAG - Use cJTAG interface
Target Interface Type [RISC-V] `target_interface_type` – Enumeration	Specifies the type of interface the target has. The options are: • JTAG - Use JTAG interface • SWD - Use SWD interface • cJTAG - Use cJTAG interface

Loader

Property	Description
Additional Load File Address[0] `debug_additional_load_file_address` – String	The address to load the additional load file.
Additional Load File Address[1] `debug_additional_load_file_address1` – String	The address to load the additional load file.
Additional Load File Address[2] `debug_additional_load_file_address2` – String	The address to load the additional load file.
Additional Load File Address[3] `debug_additional_load_file_address3` – String	The address to load the additional load file.
Additional Load File Type[0] `debug_additional_load_file_type` – Enumeration	The file type of the additional load file. The options are **Detect, elf, bin, ihex, hex, tihex, srec**.
Additional Load File Type[1] `debug_additional_load_file_type1` – Enumeration	The file type of the additional load file. The options are **Detect, elf, bin, ihex, hex, tihex, srec**.
Additional Load File Type[2] `debug_additional_load_file_type2` – Enumeration	The file type of the additional load file. The options are **Detect, elf, bin, ihex, hex, tihex, srec**.
Additional Load File Type[3] `debug_additional_load_file_type3` – Enumeration	The file type of the additional load file. The options are **Detect, elf, bin, ihex, hex, tihex, srec**.
Additional Load File[0] `debug_additional_load_file` – ProjFileName	Additional file to load on debug load. This property will have macro expansion applied to it.
Additional Load File[1] `debug_additional_load_file1` – ProjFileName	Additional file to load on debug load. This property will have macro expansion applied to it.
Additional Load File[2] `debug_additional_load_file2` – ProjFileName	Additional file to load on debug load. This property will have macro expansion applied to it.

Additional Load File[3] `debug_additional_load_file3` – ProjFileName	Additional file to load on debug load. This property will have macro expansion applied to it.
Check Load Sections Fit Target Description `target_check_load_sections_fit` – Boolean	Specifies whether load sections in the program match the memory segments described in the memory map.
Load ELF Address Limit `debug_load_file_offset_limit` – String	Restrict the Load ELF Offset.The Load ELF Offset will not be added to addresses greater than or equal to this address.
Load ELF Offset `debug_load_file_offset` – String	The offset to add to the load addresses of the ELF programs.This offset is added to any absolute relocations of symbols (whose address is less than Load ELF Offset Limit) if the load file contains relocation sections.
Load ELF Sections `debug_load_sections` – Enumeration	The debugger will load ELF sections rather than ELF programs.
Load File `external_build_file_name` – ProjFileName	The name of the main load file. This property will have macro expansion applied to it. If it is not defined then the output filepath of the linker command is used.
Load File Address `external_load_address` – String	The address to download the main load file to.
Load File Type `external_load_file_type` – Enumeration	The file type of the main load file. The options are **Detect, elf, bin, ihex, hex, tihex, srec**.
No Load Sections `target_loader_no_load_sections` – StringList	Names of (loadable) program sections or names of memory segments not to load.

ST-Link

Property	**Description**
Connect With Reset `arm_target_connect_with_reset_stlink` – Boolea	Hold the target in hardware reset on connect and stops the target. This requires the nSRST signal to be connected and the target debug hardware to work when in reset.
Host Connection `STLinkConnection` – Enumeration	A number specifying the device to connect to.
Loader Stack Size `arm_stldr_stack_size` – Integer	Specifies the stack size the STLDR loader will use
Mass Erase `arm_target_mass_erase` – Boolean	Use Mass Erase rather than sector erase on flash programming.
Primary Loader Path `arm_stldr_loader_path` – FileName	The file name of the primary loader, if empty the identified device internal flash loader is used.

ST-Link Target Interface Type `arm_target_interface_type_stlink` – Enumeratior	Specifies the type of interface the target has. The options are: • JTAG - Use JTAG interface • SWD - Use SWD interface
STM32CubeProgrammer Directory `arm_stm32cubeprogrammer_directory` – DirPath	Specifies the installation directory of the STM32CubeProgrammer software.
Secondary Loader Path `arm_stldr_loader_path2` – FileName	The file name of the secondary loader, this loader is used for address ranges that are not handled by the primary loader.
Speed `arm_target_interface_speed_stlink` – IntegerRan	The required JTAG/SWD clock frequency in kHz.

Simulator [ARM]

Property	Description
Max Instructions `arm_simulator_max_instructions` – String	Maximum number of instructions to execute before simulator is stopped.
Memory Simulation File `arm_simulator_memory_simulation_filename` – F	Specifies the dll that simulates the memory system. This property will have macro expansion applied to it. If not specified then the default memory simulation will be used.
Memory Simulation Half Words `arm_simulator_memory_half_words` – StringList	Memory half words to simulate specified by a list of address=value expressions e.g. 0xE0082002=0xface;SystemInit=0x4770
Memory Simulation Parameter `arm_simulator_memory_simulation_parameter` –	Parameter passed to the memory simulation. This property will have macro expansion applied to it.The format of this is specific to the memory simulation. The default memory simulation takes a list of **RX|RWX 'hex start address', 'hex size in bytes', 'default hex word value'** for example RX 00000000, 10000000, FFFFFFFF;RWX 10000000, 10000000, CDCDCDCD.
Memory Simulation Parameter Macros `arm_simulator_memory_simulation_parameter_`	Macros to apply to the parameter passed to the memory simulation on creation. If null then the macro MemorySegments is set to the value of the address ranges specified by the project.
Memory Simulation Words `arm_simulator_memory_words` – StringList	Memory words to simulate specified by a list of address=value expressions e.g. 0xE0082000=0xfeedface;SystemInit=0xE12FFF1E
SP Lower `arm_simulator_SP_lower` – String	Stop when R13 value is less than the value of this expression.
SP Upper `arm_simulator_SP_upper` – String	Stop when R13 value is greater than the value of this expression.

Stop On Branch . `arm_simulator_stop_on_branch_dot` – Boolean	Stop when the simulator executes a b . instruction.
Stop On Memory Error `arm_simulator_stop_on_read_write` – Enumeratio	Specifies the simulator behaviour when a memory error occurs.
Trace Buffer Size `arm_simulator_num_trace_entries` – Integer	The number of trace entries to store.

Simulator [RISC-V]

Property	Description
32-bit Divide Instruction Cost `riscv_simulator_32bit_divide_cost` – Integer	The number of cycles taken for a 32-bit divide instruction.
32-bit Multiply Instruction Cost `riscv_simulator_32bit_multiply_cost` – Integer	The number of cycles taken for a 32-bit multiply instruction.
64-bit Divide Instruction Cost `riscv_simulator_64bit_divide_cost` – Integer	The number of cycles taken for a 64-bit divide instruction.
64-bit Multiply Instruction Cost `riscv_simulator_64bit_multiply_cost` – Integer	The number of cycles taken for a 64-bit multiply instruction.
Count Cycles `riscv_simulator_count_cycles` – Boolean	Display cycle counts rather than instruction counts.
Enable Timer Support `riscv_simulator_support_mtime` – Boolean	Enable simulation of mtime and mtimecmp memory mapped registers.
Max Instructions `riscv_simulator_max_instructions` – String	Maximum number of instructions to execute before simulator is stopped.
Memory Simulation Half Words `riscv_simulator_memory_half_words` – StringList	Memory half words to simulate specified by a list of address=value expressions e.g. 0xE0082002=0xface;SystemInit=0x8082
Memory Simulation Words `riscv_simulator_memory_words` – StringList	Memory words to simulate specified by a list of address=value expressions e.g. 0xE0082000=0xfeedface;SystemInit=0x00008067
SP Lower `riscv_simulator_SP_lower` – String	Stop when x2 value is less than the value of this expression.
SP Upper `riscv_simulator_SP_upper` – String	Stop when x2 value is greater than the value of this expression.
Software Interrupt Register Address `riscv_software_interrupt_address` – IntegerHex	The address of the msip register
Stop On Jump . `riscv_simulator_stop_on_jump_dot` – Boolean	Stop when the simulator executes a j . instruction.
Timer Comparator Register Address `riscv_simulator_timer_comparator_address` – I	The address of the mtimecmp register

Timer Register Address `riscv_simulator_timer_address` – IntegerHex	The address of the mtime register
Trap On Unaligned Memory Access. `riscv_simulator_trap_on_unaligned` – Boolean	Trap on unaligned memory access.

Target Script

Property	Description
Attach Script `target_attach_script` – JavaScript	The script that is executed when the target is attached to.
Debug Begin Script `target_debug_begin_script` – JavaScript	The script that is executed when the debugger begins a debug session.
Debug End Script `target_debug_end_script` – JavaScript	The script that is executed when the debugger ends a debug session.
Load Begin Script `target_load_begin_script` – JavaScript	The script that is executed when the debugger begins a load.
Load End Script `target_load_end_script` – JavaScript	The script that is executed when the debugger ends a load.
Reset Script `target_reset_script` – JavaScript	The script that is executed when the target is reset.
Target Script File `target_script_file` – FileName	The target script file, the contents of this file are prepended to script project properties before they are executed.

Target Trace

Property	Description
ITM Stimulus Port To Display `arm_target_itm_stimulus_port_display` – Intege	Specifies the ITM Stimulus port to display in the debug terminal -1 disables this
ITM Stimulus Ports Enable `arm_target_itm_stimulus_port_enable` – IntegerH	Specifies the ITM Stimulus ports to enable.
ITM Stimulus Ports Privilege `arm_target_itm_stimulus_port_privilege` – Inte	Specifies the ITM Stimulus ports to enable.
ITM Timestamping `arm_target_itm_timestamping_enable` – Enumerat	Specifies ITM timestamping. The options are: • Disable - disable timestamping • Local - use the local timestamp clock • Global - use the global timestamp clock
ITM/DWT Data Trace PC `arm_target_dwt_data_trace_PC` – Boolean	Specifies whether to trace the PC on data trace.

ITM/DWT PC Sampling `arm_target_dwt_PC_sampling_enable` – Enumerati	Specifies the DWT PC sampling rate.
ITM/DWT Trace Exceptions `arm_target_dwt_trace_exceptions` – Boolean	Specifies whether to trace exception entry and return.
MTB RAM Address `arm_target_mtb_ram_address` – IntegerHex	Specifies the MTB RAM Address - note that this must be aligned to the MTB RAM size.
MTB RAM Size `arm_target_mtb_ram_size` – Enumeration	Specifies the MTB RAM size in bytes.
SWO Baud Rate `arm_target_trace_SWO_speed` – IntegerRange	Specifies the baud rate of the SWO - zero selects auto detection.
Trace Clock Speed `arm_target_trace_clock_speed` – IntegerRange	The speed of the trace clock. This is usually the same as the CPU clock and is used to program the prescaler for the SWO
Trace Interface Type `arm_target_trace_interface_type` – Enumeration	Specifies the type of trace interface the target has. The options are: • SWO - Use asynchronous SWO trace interface. • TracePort - Use synchronous parallel trace interface. • ETB - Use on-chip embedded trace buffer. • MTB - Use on-chip MTB - Cortex-M0+ only. • None
Trace Port Size `arm_target_trace_port_size` – Enumeration	Specifies the trace port size the target has. The options are: • 1-bit • 2-bit • 4-bit • 8-bit • 16-bit • 24-bit • 32-bit

System Macros

System Macro Values

Property	Description
$(Date) `$(Date)` – String	Day Month Year e.g. 21 June 2011.
$(DateDay) `$(DateDay)` – String	Day e.g. 21.
$(DateMonth) `$(DateMonth)` – String	Month e.g. 01 to 12.
$(DateYear) `$(DateYear)` – String	Year e.g. 2011.
$(DesktopDir) `$(DesktopDir)` – String	Path to users desktop directory.
$(DocumentsDir) `$(DocumentsDir)` – String	Path to users documents directory.
$(EmbeddedStudioVersionNumber) `$(EmbeddedStudioVersionNumber)` – String	The release version number.
$(HomeDir) `$(HomeDir)` – String	Path to users home directory.
$(HostArch) `$(HostArch)` – String	The CPU architecture that SEGGER Embedded Studio is running on e.g. x86.
$(HostArchClass) `$(HostArchClass)` – String	The class of CPU architecture that SEGGER Embedded Studio is running on e.g. intel, arm.
$(HostDLL) `$(HostDLL)` – String	The file extension for dynamic link libraries on the CPU that SEGGER Embedded Studio is running on e.g. .dll.
$(HostDLLExt) `$(HostDLLExt)` – String	The file extension for dynamic link libraries used by the operating system that SEGGER Embedded Studio is running on e.g. .dll, .so, .dylib.
$(HostEXE) `$(HostEXE)` – String	The file extension for executables on the CPU that SEGGER Embedded Studio is running on e.g. .exe.
$(HostOS) `$(HostOS)` – String	The name of the operating system that SEGGER Embedded Studio is running on e.g. win.
$(JLinkDir) `$(JLinkDir)` – String	Path to J-Link software.
$(Micro) `$(Micro)` – String	The SEGGER Embedded Studio target e.g. ARM.

$(PackagesDir) `$(PackagesDir)` – String	Path to the users packages directory.
$(Platform) `$(Platform)` – String	The target platform.
$(ProductNameShort) `$(ProductNameShort)` – String	The product name.
$(StudioArchiveFileExt) `$(StudioArchiveFileExt)` – String	The filename extension of a studio archive file.
$(StudioBuildToolExeName) `$(StudioBuildToolExeName)` – String	The filename of the build tool executable.
$(StudioBuildToolName) `$(StudioBuildToolName)` – String	The name of the build tool executable.
$(StudioDir) `$(StudioDir)` – String	The install directory of the product.
$(StudioExeName) `$(StudioExeName)` – String	The filename of the studio executable.
$(StudioLicenseToolExeName) `$(StudioLicenseToolExeName)` – String	The filename of the license tool executable.
$(StudioLicenseToolName) `$(StudioLicenseToolName)` – String	The name of the license tool executable.
$(StudioMajorVersion) `$(StudioMajorVersion)` – String	The major release version of software.
$(StudioMinorVersion) `$(StudioMinorVersion)` – String	The minor release version of software.
$(StudioName) `$(StudioName)` – String	The full name of studio.
$(StudioNameShort) `$(StudioNameShort)` – String	The short name of studio.
$(StudioPackageFileExt) `$(StudioPackageFileExt)` – String	The filename extension of a studio package file.
$(StudioProjectFileExt) `$(StudioProjectFileExt)` – String	The filename extension of a studio project file.
$(StudioScriptToolExeName) `$(StudioScriptToolExeName)` – String	The filename of the script tool executable.
$(StudioScriptToolName) `$(StudioScriptToolName)` – String	The name of the script tool executable.
$(StudioSessionFileExt) `$(StudioSessionFileExt)` – String	The filename extension of a studio session file.
$(StudioSimulatorExeName) `$(StudioSimulatorExeName)` – String	The filename of the simulator executable.

$(StudioSimulatorName) `$(StudioSimulatorName)` – String	The name of the simulator executable.
$(StudioUserDir) `$(StudioUserDir)` – String	The directory containing the user data.
$(TargetID) `$(TargetID)` – String	ID number representing the SEGGER Embedded Studio target.
$(Time) `$(Time)` – String	Hour:Minutes:Seconds e.g. 15:34:03.
$(TimeHour) `$(TimeHour)` – String	Hour e.g. 15.
$(TimeMinute) `$(TimeMinute)` – String	Minute e.g. 34.
$(TimeSecond) `$(TimeSecond)` – String	Seconds e.g. 03.
$(UnixTime) `$(UnixTime)` – String	Seconds since 00:00, Jan 1 1970 UTC

Build Macros

(Build Macro Values)

Property	Description
$(ABI) `$(ABI)` – String	The value of the **RISC-V ABI Type** project property.
$(AR) `$(AR)` – String	The path to the binutils ar command.
$(AS) `$(AS)` – String	The path to the binutils as command.
$(Arch) `$(Arch)` – String	The lower case value of the **ARM Architecture** project property.
$(Arch) `$(Arch)` – String	The value of the **RISC-V ISA** project property.
$(Arch2) `$(Arch2)` – String	Alternative value of the **ARM Architecture** project property.
$(AsmOptions) `$(AsmOptions)` – String	A space seperated list of assembler options for the external assemble command.
$(CC) `$(CC)` – String	The path to the cc command.
$(CC1) `$(CC1)` – String	The path to the gcc cc1 command.
$(CCPP) `$(CCPP)` – String	The path to the cc command.
$(CLANG) `$(CLANG)` – String	The path to the clang command.
$(CLANGTIDY) `$(CLANGTIDY)` – String	The path to the clang-tidy command.
$(COnlyOptions) `$(COnlyOptions)` – String	A space seperated list of compiler options for the external c compile command.
$(COptions) `$(COptions)` – String	A space seperated list of compiler options for the external c and c++ compile commands.
$(CombiningOutputFilePath) `$(CombiningOutputFilePath)` – String	The full path of the output file of the combining command.
$(CombiningRelInputPaths) `$(CombiningRelInputPaths)` – String	The relative inputs to the combining command.
$(Configuration) `$(Configuration)` – String	The build configuration e.g. ARM Flash Debug.

$(ConfigurationHash) `$(ConfigurationHash)` – String	The hash value of build configuration.
$(CoreType) `$(CoreType)` – String	The lower case value of the **ARM Core Type** project property.
$(Defines) `$(Defines)` – String	The preprocessor defines property value for the external compile command.
$(DependencyPath) `$(DependencyPath)` – String	The path of the dependency file for the external compile command.
$(EXE) `$(EXE)` – String	The default file extension for an executable file including the dot e.g. .elf.
$(Endian) `$(Endian)` – String	The lower case value of the **Byte Order** project property.
$(FPABI) `$(FPABI)` – String	The value of the **ARM FP ABI Type** project property.
$(FPU) `$(FPU)` – String	The lower case value of the **ARM FPU Type** project property.
$(FPU2) `$(FPU2)` – String	Alternative value of the **ARM FPU Type** project property.
$(FPU3) `$(FPU3)` – String	Alternative value of the **ARM FPU Type** project property.
$(FolderName) `$(FolderName)` – String	The folder name of the containing folder.
$(FolderPath) `$(FolderPath)` – String	The folder path of the containing folders.
$(GCC) `$(GCC)` – String	The path to the gcc command.
$(GCCPrefix) `$(GCCPrefix)` – String	The macro-expanded value of the **GCC Prefix** project property.
$(GCCTarget) `$(GCCTarget)` – String	The macro-expanded value of the **GCC Target** project property.
$(GCCVersion) `$(GCCVersion)` – String	The macro-expanded value of the **GCC Version** project property.
$(GPLUSPLUS) `$(GPLUSPLUS)` – String	The path to the g++ command.
$(IncludeFiles) `$(IncludeFiles)` – String	The user includes property value for the external compile command.
$(Includes) `$(Includes)` – String	The user directories property value for the external compile command.
$(InputDir) `$(InputDir)` – String	The absolute directory of the input file.

$(InputExt) `$(InputExt)` – String	The extension of an input file not including the dot e.g cpp.
$(InputFileName) `$(InputFileName)` – String	The name of an input file relative to the project directory.
$(InputName) `$(InputName)` – String	The name of an input file relative to the project directory without the extension.
$(InputPath) `$(InputPath)` – String	The absolute name of an input file including the extension.
$(InputPathHash) `$(InputPathHash)` – String	The hash value of the absolute name of an input file including the extension.
$(IntDir) `$(IntDir)` – String	The macro-expanded value of the **Intermediate Directory** project property.
$(LD) `$(LD)` – String	The path to the binutils ld command.
$(LIB) `$(LIB)` – String	The default file extension for a library file including the dot e.g. .lib.
$(LIBLTO) `$(LIBLTO)` – String	The path to the LTO dll.
$(LTO1) `$(LTO1)` – String	The path to the gcc lto1 command.
$(LibArch) `$(LibArch)` – String	The library architecture.
$(LibEndianExt) `$(LibEndianExt)` – String	The endian specific library extension.
$(LibExt) `$(LibExt)` – String	The architecture and build specific library extension.
$(LinkLibraries) `$(LinkLibraries)` – String	The value of the **Standard Libraries Directory** project property.
$(LinkOptions) `$(LinkOptions)` – String	A space seperated list of compiler options for the external link command.
$(LinkerScriptPath) `$(LinkerScriptPath)` – String	The full path of the linker script file for the link command.
$(MacroFiles) `$(MacroFiles)` – String	The user macros property value for the external compile command.
$(MapPath) `$(MapPath)` – String	The full path of the map file of the external link command.
$(MemorySegments) `$(MemorySegments)` – String	The value of the Memory Segments property supplied to pre/post link command.
$(OBJ) `$(OBJ)` – String	The default file extension for an object file including the dot e.g. .o.

$(OBJCOPY) `$(OBJCOPY)` – String	The path to the binutils objcopy command.
$(OBJDUMP) `$(OBJDUMP)` – String	The path to the binutils objdump command.
$(Objects) `$(Objects)` – String	A space seperated list of files for the external archive or link command.
$(ObjectsFilePath) `$(ObjectsFilePath)` – String	The full path containing the files for the external archive or link command.
$(OutDir) `$(OutDir)` – String	The macro-expanded value of the **Output Directory** project property.
$(PackageExt) `$(PackageExt)` – String	The file extension of a package file e.g. emPackage.
$(PostLinkOutputFilePath) `$(PostLinkOutputFilePath)` – String	The full path of the output file of the pre link command.
$(PostLinkOutputFilePath) `$(PostLinkOutputFilePath)` – String	The full path of the output file of the post link command.
$(PostLinkOutputFilePath) `$(PostLinkOutputFilePath)` – String	The full path of the output file of the pre link command.
$(PostLinkOutputFilePath) `$(PostLinkOutputFilePath)` – String	The full path of the output file of the post link command.
$(ProjectDir) `$(ProjectDir)` – String	The absolute value of the **Project Directory** project property of the current proje ct. If this isn't set then the directory containing the solution file.
$(ProjectName) `$(ProjectName)` – String	The project name of the current project.
$(ProjectNodeName) `$(ProjectNodeName)` – String	The name of the selected project node.
$(RANLIB) `$(RANLIB)` – String	The path to the binutils ranlib command.
$(RelDependencyPath) `$(RelDependencyPath)` – String	The relative path of the dependency file for the external compile command.
$(RelInputDir) `$(RelInputDir)` – String	The relative path to the directory containing the input file from the project directory or dot if not relative.
$(RelInputPath) `$(RelInputPath)` – String	The relative path to the input file from the project directory or the full path if not relative.
$(RelLinkerScriptPath) `$(RelLinkerScriptPath)` – String	The relative path of the linker script file for the link command.
$(RelMapPath) `$(RelMapPath)` – String	The relative path of the map file of the external link command.
$(RelObjectsFilePath) `$(RelObjectsFilePath)` – String	The relative path containing the files for the external archive or link command.

$(RelTargetPath) `$(RelTargetPath)` – String	The project directory relative path of the output file of the link or compile command.
$(RootOutDir) `$(RootOutDir)` – String	The macro-expanded value of the **Root Output Directory** project property.
$(RootRelativeOutDir) `$(RootRelativeOutDir)` – String	The relative path to get from the path specified by the **Output Directory** project property to the path specified by the **Root Output Directory** project property.
$(SASM) `$(SASM)` – String	The path to the SEGGER assembler.
$(SCC) `$(SCC)` – String	The path to the SEGGER compiler.
$(SLINK) `$(SLINK)` – String	The path to the SEGGER linker.
$(STRIP) `$(STRIP)` – String	The path to the binutils strip command.
$(SolutionDir) `$(SolutionDir)` – String	The absolute path of the directory containing the solution file.
$(SolutionExt) `$(SolutionExt)` – String	The extension of the solution file without the dot.
$(SolutionFileName) `$(SolutionFileName)` – String	The filename of the solution file.
$(SolutionName) `$(SolutionName)` – String	The basename of the solution file.
$(SolutionPath) `$(SolutionPath)` – String	The absolute path of the solution file.
$(StageOutputFilePath) `$(StageOutputFilePath)` – String	The full path of the output file of the stage command.
$(TargetPath) `$(TargetPath)` – String	The full path of the output file of the link or compile command.
$(ToolChainDir) `$(ToolChainDir)` – String	The macro-expanded value of the **Tool Chain Directory** project property.
$(Undefines) `$(Undefines)` – String	The preprocessor undefines property value for the external compile command.

BinaryFile

The following table lists the BinaryFile object's member functions.

BinaryFile.crc32(offset, length) returns the CRC-32 checksum of an address range *length* bytes long, starting at *offset*. This function computes a CRC-32 checksum on a block of data using the standard CRC-32 polynomial (0x04C11DB7) with an initial value of 0xFFFFFFFF. Note that this implementation doesn't reflect the input or the output and the result is inverted.
BinaryFile.length() returns the length of the binary file in bytes.
BinaryFile.load(path) loads binary file from *path*.
BinaryFile.loadAppend(path) loads binary file from *path* and appends it to the binary image.
BinaryFile.peekBytes(offset, length) returns byte array containing *length* bytes peeked from *offset*.
BinaryFile.peekUint32(offset, littleEndian) returns a 32-bit word peeked from *offset*. The *littleEndian* argument specifies the endianness of the access, if true or undefined it will be little endian, otherwise it will be big endian.
BinaryFile.pokeBytes(offset, byteArray) poke byte array *byteArray* to *offset*.
BinaryFile.pokeUint32(offset, value, littleEndian) poke a *value* to 32-bit word located at *offset*. The *littleEndian* argument specifies the endianness of the access, if true or undefined it will be little endian, otherwise it will be big endian.
BinaryFile.resize(length, fill) resizes the binary image to *length* bytes. If the operation extends the size, the binary image will be padded with bytes of value *fill*.
BinaryFile.save(path) saves binary file to *path*.
BinaryFile.saveRange(path, offset, length) saves part of the binary file to *path*. The *offset* argument specifies the byte offset to start from. The *length* argument specifies the maximum number of bytes that should be saved.

CWSys

The following table lists the CWSys object's member functions.

CWSys.appendStringToFile(path, string) appends *string* to the end of the file *path*.
CWSys.copyFile(srcPath, destPath) copies file *srcPath* to *destPath*.
CWSys.crc32(array) returns the CRC-32 checksum of the byte array *array*. This function computes a CRC-32 checksum on a block of data using the standard CRC-32 polynomial (0x04C11DB7) with an initial value of 0xFFFFFFFF. Note that this implementation doesn't reflect the input or the output and the result is inverted.
CWSys.fileExists(path) returns true if file *path* exists.
CWSys.fileSize(path) return the number of bytes in file *path*.
CWSys.getRunStderr() returns the stderr output from the last *CWSys.run()* call.
CWSys.getRunStdout() returns the stdout output from the last *CWSys.run()* call.
CWSys.makeDirectory(path) create the directory *path*.
CWSys.packU32(array, offset, number, le) packs *number* into the *array* at *offset*.
CWSys.popup(text, caption) prompt the user with text and return true for yes and false for no.
CWSys.readByteArrayFromFile(path) returns the byte array contained in the file *path*.
CWSys.readStringFromFile(path) returns the string contained in the file *path*.
CWSys.removeDirectory(path) remove the directory *path*.
CWSys.removeFile(path) deletes file *path*.
CWSys.renameFile(oldPath, newPath) renames file *oldPath* to be *newPath*.
CWSys.run(cmd, wait) runs command line *cmd* optionally waits for it to complete if *wait* is true.
CWSys.unpackU32(array, offset, le) returns the number unpacked from the *array* at *offset*.
CWSys.writeByteArrayToFile(path, array) creates a file *path* containing the byte array *array*.
CWSys.writeStringToFile(path, string) creates a file *path* containing *string*.

Debug

The following table lists the Debug object's member functions.

Debug.evaluate(expression) evaluates debug *expression* and returns it as a JavaScript value.
Debug.getfunction(address) return function name containing *address*.

ElfFile

The following table lists the ElfFile object's member functions.

ElfFile.crc32(address, length, virtualNotPhysical, padding, programNotSection) returns the CRC-32 checksum of an address range *length* bytes long, located at *address*. If *virtualNotPhysical* is true or undefined, *address* is a virtual address otherwise it is a physical address. If *padding* is defined, it specifies the byte value used to fill gaps in the program. If *programNotSection* is true or undefined, data is read using program headers rather than section headers. This function computes a CRC-32 checksum on a block of data using the standard CRC-32 polynomial (0x04C11DB7) with an initial value of 0xFFFFFFFF. Note that this implementation doesn't reflect the input or the output and the result is inverted.
ElfFile.findProgram(address) returns an object with *start*, the *data* and the *size* to allocate of the Elf program that contains *address*.
ElfFile.getEntryPoint() returns the entry point in the ELF file.
ElfFile.getSection(name) returns an object with *start* and the *data* of the Elf section corresponding to the *name*.
ElfFile.isLittleEndian() returns true if the Elf file has numbers encoded as little endian.
ElfFile.load(path) loads Elf file from *path*.
ElfFile.peekBytes(address, length, virtualNotPhysical, padding, programNotSection) returns byte array containing *length* bytes peeked from *address*. If *virtualNotPhysical* is true or undefined, *address* is a virtual address otherwise it is a physical address. If *padding* is defined, it specifies the byte value used to fill gaps in the program. If *programNotSection* is true or undefined, data is read using program headers rather than section headers.
ElfFile.peekUint32(address, virtualNotPhysical) returns a 32-bit word peeked from *address*. If *virtualNotPhysical* is true or undefined, *address* is a virtual address otherwise it is a physical address.
ElfFile.pokeBytes(address, byteArray, virtualNotPhysical) poke byte array *byteArray* to *address*. If *virtualNotPhysical* is true or undefined, *address* is a virtual address otherwise it is a physical address.
ElfFile.pokeUint32(address, value, virtualNotPhysical) poke a *value* to 32-bit word located at *address*. If *virtualNotPhysical* is true or undefined, *address* is a virtual address otherwise it is a physical address.
ElfFile.save(path) saves Elf file to *path*.
ElfFile.symbolValue(symbol) returns the value of *symbol* in Elf file.

TargetInterface

The following table lists the TargetInterface object's member functions.

TargetInterface.crc32(address, length) reads a block of bytes from target memory starting at address for length bytes, generates a crc32 on the block of bytes and returns it.
TargetInterface.delay(ms) waits for ms milliseconds
TargetInterface.error(message) terminates execution of the script and outputs a target interface error message to the target log.
TargetInterface.executeFunction(address, parameter, timeout) calls a function at address with the parameter and returns the function result. The timeout is in milliseconds.
TargetInterface.expandMacro(string) returns the string with macros expanded.
TargetInterface.findByte(address, length, byte) returns the index of the byte in the specified target memory range.
TargetInterface.findNotByte(address, length, byte) returns the index of the byte that isn't in the specified target memory range.
TargetInterface.getProjectProperty(savename) returns the value of the savename project property.
TargetInterface.getTargetProperty(savename) returns the value of the savename target property.
TargetInterface.go() allows the target to run.
TargetInterface.isStopped() returns true if the target is stopped.
TargetInterface.message(message) outputs a target interface message to the target log.
TargetInterface.peekBinary(address, length, filename) reads a block of bytes from target memory starting at address for length bytes and writes them to filename.
TargetInterface.peekByte(address) reads a byte of target memory from address and returns it.
TargetInterface.peekBytes(address, length) reads a block of bytes from target memory starting at address for length bytes and returns the result as an array containing the bytes read.
TargetInterface.peekMultUint16(address, length) reads length unsigned 16-bit integers from target memory starting at address and returns them as an array.
TargetInterface.peekMultUint32(address, length) reads length unsigned 32-bit integers from target memory starting at address and returns them as an array.
TargetInterface.peekUint16(address) reads a 16-bit unsigned integer from target memory from address and returns it.
TargetInterface.peekUint32(address) reads a 32-bit unsigned integer from target memory from address and returns it.
TargetInterface.peekWord(address) reads a word as an unsigned integer from target memory from address and returns it.
TargetInterface.pokeBinary(address, filename) reads a block of bytes from filename and writes them to target memory starting at address.
TargetInterface.pokeByte(address, data) writes the byte data to address in target memory.

TargetInterface.pokeBytes(address, data) writes the array data containing 8-bit data to target memory at address.

TargetInterface.pokeMultUint16(address, data) writes the array data containing 16-bit data to target memory at address.

TargetInterface.pokeMultUint32(address, data) writes the array data containing 32-bit data to target memory at address.

TargetInterface.pokeUint16(address, data) writes data as a 16-bit value to address in target memory.

TargetInterface.pokeUint32(address, data) writes data as a 32-bit value to address in target memory.

TargetInterface.pokeWord(address, data) writes data as a word value to address in target memory.

TargetInterface.readBinary(filename) reads a block of bytes from filename and returns them in an array.

TargetInterface.reset() resets the target.

TargetInterface.resetAndStop() resets and stops the target.

TargetInterface.runFromAddress(address, timeout) start the target executing at address and waits for a breakpoint to be hit. The timeout is in milliseconds.

TargetInterface.runFromToAddress(from, to, timeout) start the target executing at address from and waits for the breakpoint to be hit. The timeout is in milliseconds.

TargetInterface.runToAddress(address, timeout) sets a breakpoint at address, starts the target executing and waits for the breakpoint to be hit. The timeout is in milliseconds.

TargetInterface.setTargetProperty(savename) set the value of the savename target property.

TargetInterface.stop() stops the target.

TargetInterface.writeBinary(array, filename) write the bytes in array to filename.

WScript

The following table lists the WScript object's member functions.

WScript.Echo(s) echos string *s* to the output terminal.